UNIVERSIDAD DE MÁLAGA

E.T.S. INGENIERÍA INFORMÁTICA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA GRADUADO EN INGENIERÍA DEL SOFTWARE

Desarrollo de sistema para caracterizar ruidos urbanos y transformarlos a otro tipo de sonidos agradables al oído humano

Development of a system to characterize urban noises and transform them into other types of pleasant sounds to the human ear

Realizado por

José Santos Garrido

Tutorizado por

Francisco Vico Vela

Departamento

Lenguajes y Ciencias de la Computación

UNIVERSIDAD DE MÁLAGA

MÁLAGA, JUNIO DE 2019

Fecha defensa: de julio de 2019

Fdo. El/la Secretario/a del Tribunal

Resumen

Este TFG pretende diseñar e implementar una aplicación software capaz de organizar el sonido ambiente captado por el micrófono del dispositivo. Para ello el sistema tendrá que analizar y caracterizar el sonido mediante el uso de técnicas procedentes del análisis de señal complementado con inteligencia artificial. Ya analizado y caracterizado, el sistema, basándose en conocimientos sobre el análisis musical y los principios fundamentales de la melodía, la armonía y el ritmo, el sistema aplicará una serie de reglas musicales y cambios de timbres con el fin de obtener y reproducir un sonido más estructurado y melodioso. Este sonido resultante tendrá, por lo tanto, una fuerte relación con el audio anteriormente adquirido por el dispositivo. Además, dicho sistema contará con la implementación de un reproductor MIDI y WAV, ya que no todos los sistemas operativos ofrecen uno por defecto que pueda reproducir los dos formatos y la aplicación trabaja con ambos. En adición, ofrece cierta personalización para que el sonido resultante intente ser acorde a los gustos del usuario.

Palabras clave: MIDI, WAV, Audio, Sonido, Señal, Música, Ruido

Abstract

This TFG aims to design and implement a software application capable of organizing the ambient sound captured by the microphone of the device. For this the system will have to analyze and characterize the sound through the use of techniques from signal analysis supplemented with artificial intelligence. Already analyzed and characterized, the system, based on knowledge about musical analysis and the fundamental principles of melody, harmony and rhythm, the system will apply a series of musical rules and changes of timbres in order to obtain and reproduce a sound more structured and melodious. This resulting sound will, therefore, have a strong relationship with the audio previously acquired by the device. In addition, this system will have the implementation of a MIDI and WAV player, since not all operating systems offer one by default that can reproduce the two formats and the application works with both. In addition, it offers some customization so that the resulting sound tries to be according to the user's tastes.

Keywords: MIDI, WAV, Audio, Sound, Signal, Music, Noise

Índice

INTRODUCCIÓN 1

1.1 MOTIVACIÓN 1
1.2 OBJETIVOS 2
1.3 ESTRUCTURA DE LA MEMORIA 2
1.4 TECNOLOGÍAS UTILIZADAS 3
1.4.1 Python 3
1.4.2 C# 4
1.4.3 JSON 6
1.4.4 Microsoft .NET 7
1.4.5 Unity 8
1.4.6 Mono 10
1.4.7 Microsoft Visual Studio 11
1.4.8 Geany 12
1.4.9 Essentia 12
1.4.10 Accord.NET 13
1.4.11 NAudioUnity 13
1.4.12 DryWetMidi 14
1.4.13 UnitySimpleFileBrowser 14
1.4.14 MidiPlayerToolKitUnity 15
1.4.15 JSON.NET 15
1.4.16 Metodología ágil 16
1.4.17 Programación Extrema Personal 16

METODOLOGÍA 19

2.1 PROGRAMACIÓN EXTREMA PERSONAL 19

REQUISITOS 23

3.1 ANÁLISIS DE LOS REQUISITOS 23
3.1.1 Requisitos funcionales 23
3.1.2 Requisitos no funcionales 25

PREPARACIÓN DEL DESARROLLO 27

4.1 OBTENCIÓN DE LA DOCUMENTACIÓN 27

4.1.1 Acústica 28

4.1.2 Transformada de Fourier 28

4.1.3 Registro digital del sonido 30

DESARROLLO 35

5.1 ANÁLISIS Y CLASIFICACIÓN DE UN AUDIO REGISTRADO 35

5.1.1 Inicialización de la iteración 35

5.1.2 Diseño 35

5.1.3 Implementación 36

5.1.4 Pruebas del sistema 43

5.1.5 Retrospectiva 43

5.2 CLASIFICAR UN AUDIO EN DIRECTO 43

5.2.1 Inicialización de la iteración 43

5.2.2 Diseño 43

5.2.3 Implementación 44

5.2.4 Pruebas 46

5.2.5 Retrospectiva 46

5.3 ANÁLISIS MUSICAL DEL SONIDO AMBIENTE 47

5.3.1 Inicialización de la iteración 47

5.3.2 Diseño 47

5.3.3 Implementación 47

5.3.4 Pruebas 55

5.3.5 Retrospectiva 56

5.4 TRANSFORMACIÓN DEL SONIDO 56

5.4.1 Inicialización de la iteración 56

5.4.2 Diseño 56

5.4.3 Implementación 57

5.4.4 Pruebas 65

5.4.5 Retrospectiva 65

5.5 APLICACIÓN DE LOS RESULTADOS 66

5.5.1 Inicialización de la iteración 66

5.5.2 Diseño 66

5.5.3 Implementación 68
5.5.4 Pruebas 89
5.5.5 Retrospectiva 89

CONCLUSIONES 91

6.1 Conclusiones 91
6.2 Trabajo futuro 93

BIBLIOGRAFÍA 95

MANUAL DE USUARIO 97

A.1 Requisitos e instalación 97
A.2 Reproductor 98
A.3 Transformador Audio 99
A.4 Transformador Audio Stream 100
A.5 Opciones 101

Índice de Figuras

Figura 1.1: Logotipo de Python. 4
Figura 1.2: Logotipo de C#. 5
Figura 1.3: Ejemplo de documentación de C# con su compilador online. 6
Figura 1.4: Logotipo de JSON. 6
Figura 1.5: Logotipo de Microsoft .NET. 7
Figura 1.6: Logotipo de Unity. 8
Figura 1.7: Distintas plataformas exportables por Unity. 9
Figura 2.1: Esquema Personal Extreme Programming. 20
Figura 4.1: Transformada de Fourier. 28
Figura 4.2: Ejemplo de onda producida por un sintetizador. 29
Figura 4.3: Primera frecuencia obtenida de la transformada de Fourier. 29
Figura 4.4: Segunda frecuencia obtenida de la transformada de Fourier. 30
Figura 4.5: Esquema formato WAV. Soundfile. *http://soundfile.sapp.org/doc/WaveFormat/* 33
Figura 5.1: Audios de muestra ordenados. 38
Figura 5.2: Diagrama de clase Utilidades. 38
Figura 5.3: Diagrama de clase Clasificador. 40
Figura 5.4: La 440 (A4, A440, La4). 49
Figura 5.5 Diagrama de clase NotaMusical. 51
Figura 5.6: Espectro del silencio. 52
Figura 5.7: Espectro de onda senoidal a 440Hz. 53
Figura 5.8: Espectro de la nota A1 tocada en un piano. 53
Figura 5.9: Diagrama de clase FiguraMusical. 58
Figura 5.10: Diagrama de clase Composicion. 60
Figura 5.11: Diagrama de struct Frame. 61
Figura 5.12: Diagrama de clase Compositor 65
Figura 5.13: Boceto diseño interfaz de usuario. 67
Figura 5.14: Diagrama de clase Reproductor. 69
Figura 5.15: Diagrama de clase ReproductorFicheros. 70
Figura 5.16: Diagrama de clase AudioFilePlayer. 71

Figura 5.17: Diagrama de clase MidiExternalFilePlayer..... 72
Figura 5.18: Diagrama de clase ReproductorStreamMidi. 72
Figura 5.19: Diagrama de clase PanelFuncion. 74
Figura 5.20: Diagrama de clase ManagerInterfaz. 75
Figura 5.21: Diagrama de clase BotonFuncion. 75
Figura 5.22: Diagrama de clase SistemaGuardado. 76
Figura 5.23: Diagrama de clase PanelFuncionMuteable..... 76
Figura 5.24: Diagrama de clase PanelReproductor. 77
Figura 5.25: Vista de PanelReproductor. 78
Figura 5.26: Diagrama de clase PanelTransformarAudio. 80
Figura 5.27: Vista de PanelTransformarAudio. 81
Figura 5.29: Vista de PanelTransformarAudioStream. 84
Figura 5.30: Diagrama de clase PanelOpciones. 86
Figura 5.31: Diagrama de clase BotonInstrumento. 87
Figura 5.32: Diagrama de clase Link. 87
Figura 5.33: Vista de PanelOpciones. 88
Figura 5.34: Diagrama de clase ErrorManager. 88
Figura A.1: Vista primera ejecución. 98
Figura A.2: Panel Reproductor de audio. 99
Figura A.3: Panel Transformador Audio. 100
Figura A.3: Panel Transformador Audio Stream. 101
Figura A.5: Panel Opciones. 102

1

Introducción

1.1 Motivación

Actualmente, debido a la gran actividad humana, estamos expuestos diariamente a un exceso de sonido llamado contaminación acústica. Este sonido molesto puede producir efectos nocivos fisiológicos y psicológicos para las personas. Una posible solución a este problema sería aislarse acústicamente, pero esto posiblemente acarree otros inconvenientes, ya que supone anular uno de los cinco sentidos y no proveer información del alrededor, lo que podría producir, por ejemplo, atropellos, si un peatón no escucha el sonido de un vehículo. Este caso se podría solucionar sustituyendo el sonido molesto del vehículo por el de algún instrumento, con el fin de que el peatón pueda recibir los estímulos de una forma más agradable y, por lo tanto, menos perniciosa para la salud.

Este proyecto se basa en un trabajo de la Universidad de New York llamado "*A Dataset and Taxonomy for Urban Sound Research*", en dicho trabajo se estudia cómo clasificar los sonidos. Nuestra contribución será asociar y reemplazar un sonido por otro mediante el uso de dicha clasificación.

1.2 Objetivos

El objetivo de este proyecto es desarrollar una aplicación multiplataforma con el fin de convertir el sonido ambiente a otro más agradable. La aplicación necesitará dispositivos tanto para la entrada como para la salida de audio.

Será necesario adquirir y emplear diversos conocimientos sobre acústica, análisis de señal y formatos de audio para poder implementar la lógica del sistema a desarrollar.

Para la conversión del sonido, el sistema captará el audio mediante el dispositivo de grabación, posteriormente, se aplicarán algoritmos empleados en el análisis de señales. A continuación, se aplicará un análisis musical para obtener y modificar las propiedades del sonido captado para, finalmente, reproducir el audio resultante a través del dispositivo de salida.

Gracias a esto, conseguimos transformar un sonido de una fuente a otra, por ejemplo, de un perro a un piano. Además, en todo momento se mantiene una correlación entre los conjuntos de propiedades de ambos sonidos.

1.3 Estructura de la memoria

Esta memoria consta de seis capítulos contando con esta Introducción. A continuación, se explica el contenido de cada uno de los capítulos.

Introducción

En este capítulo se detallan los objetivos de este proyecto y las tecnologías empleadas en durante el desarrollo de este.

Metodología

En este capítulo se explica la metodología empleada durante el desarrollo del sistema.

Preparación del desarrollo

En este capítulo se muestran los conocimientos adquiridos y empleados para poder implementar la lógica del sistema.

Desarrollo

En este capítulo se describen cada una de las iteraciones en las que se ha dividido el proceso de desarrollo de la aplicación. Cada iteración consta de varias fases.

Conclusiones

En este capítulo se exponen las conclusiones obtenidas en el proyecto y la visión futura del mismo.

1.4 Tecnologías utilizadas

En esta sección se describen las tecnologías utilizadas para el desarrollo del proyecto. Se definirá cada tecnología y se explicarán algunos de sus aspectos fundamentales.

1.4.1 Python

Python es un lenguaje de programación interpretado, fue creado a finales de los ochenta por Guido van Rossum como un sucesor del lenguaje de programación ABC. El nombre del lenguaje proviene de la afición de su creador por los humoristas británicos Monty Python. La figura 1.1 muestra el logotipo del lenguaje.

Figura 1.1: Logotipo de Python.

Python es un lenguaje de programación multiparadigma debido a que soporta orientación a objetos y, en menor medida, programación funcional. Además, usa tipado dinámico y es multiplataforma. Es administrado por Python Software Foundation y a partir de la versión 2.1.1 es compatible con la Licencia pública general de GNU.

Python fue publicado por van Rossum en 1991 con la versión 0.9.0, en esta versión ya presentaba clases con herencia, manejo de excepciones, funciones y los tipos modulares; la versión 1.0 se alcanzó en enero de 1994 implementando características de la programación funcional.

La filosofía Python es bastante similar a la filosofía de Unix, el código debe ser legible y transparente, evitando, por lo tanto, el código opaco u ofuscado. Estos principios fueron descritos por un desarrollador de Python llamado Tim Peters en “El Zen de Python”.

Python se ha utilizado para poder trabajar con la librería Essentia (explicada más adelante), concretamente se ha trabajado con la versión 2.7.

1.4.2 C#

C# es un lenguaje de programación multiparadigma influido principalmente por C++ y Java, abarca tipografía fuerte, de ámbito léxico, imperativo, declarativo, funcional, reflexivo, genérico, concurrente, orientado a objetos y disciplinas de programación orientadas a componentes. Fue diseñado por Anders Hejlsberg, empleado de Microsoft. Microsoft lo publicó alrededor del año 2000 dentro de su

iniciativa .NET, más tarde fue aprobado como estándar por ECMA e ISO.

El nombre C# hace referencia al símbolo '++' que indica que la variable debe ser incrementada en uno después de su lectura, y como C# está basado en C++ al incrementar nuevamente el nombre de este último lenguaje nos quedaría C++++ estando esos cuatro símbolos de suma agrupados formando un sostenido. Además, hace alusión a la teoría musical, la nota Do se representa con la letra C y al añadirle el sostenido nos quedaría Do sostenido (C#), una nota un semitono más agudo, indicando, por lo tanto, que C# es una versión extendida de C.

El lenguaje C# está diseñado para generar programas sobre la plataforma .NET o plataformas compatibles a esta, como por ejemplo Mono. Además, se ha de hacer mención a la documentación de este lenguaje, siendo altamente detallada e ilustrativa mediante ejemplos e incluso con un compilador online para hacer pruebas.

Figura 1.2: Logotipo de C#.

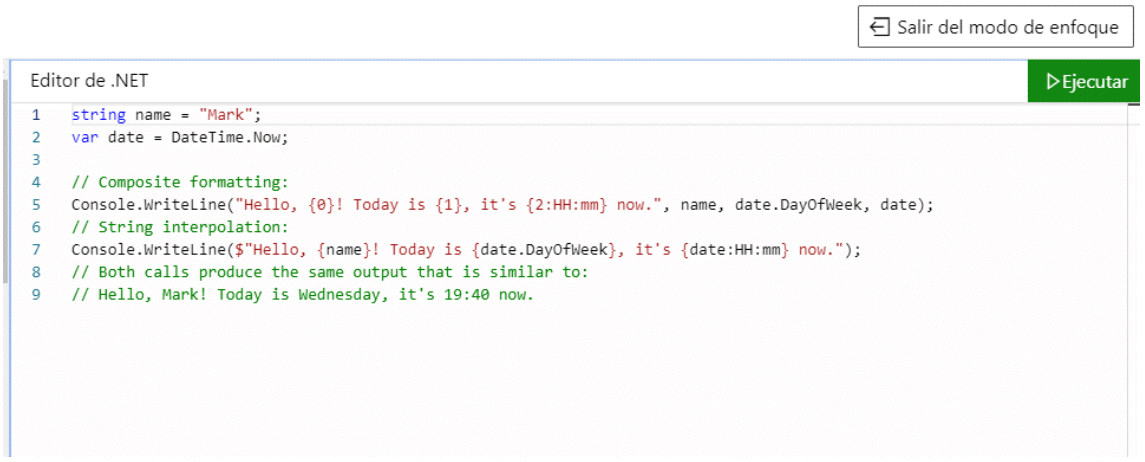

Figura 1.3: Ejemplo de documentación de C# con su compilador online.

Se ha elegido C# como lenguaje principal del proyecto debido a que se dispone de experiencia previa con esta tecnología y, además, se desea seguir aprendiendo y desarrollar habilidades con ella.

1.4.3 JSON

JSON es un formato de texto para el intercambio de datos, proviene de javascript ya que forma parte de su notación literal de objetos, aunque actualmente se considera como un formato independiente del lenguaje debido a su amplia aceptación como alternativa a XML

Figura 1.4: Logotipo de JSON.

JSON se ha utilizado para serializar y deserializar los datos de configuración del usuario de la aplicación con el fin de guardarlos en un fichero, y mejorar la experiencia del cliente.

1.4.4 Microsoft .NET

.NET es un framework de Microsoft que facilita la independencia de la plataforma hardware, la transparencia en redes y el desarrollo de aplicaciones en plataformas con sistema operativo Windows. Este framework permite la interoperabilidad entre multitud de lenguajes.
La Figura 1.5 muestra el logotipo de .NET.

Figura 1.5: Logotipo de Microsoft .NET.

Los principales componentes de Microsoft .NET son:

-Lenguajes de programación: gracias a la infraestructura común de lenguajes (CLI) .NET soporta más de 20 lenguajes, entre ellos se encuentran: C#, C/C++, Basic, Python, Cobol y Fortran.

-Biblioteca de clases base (BCL): es el componente más importante, define el conjunto funcional mínimo que debe implementarse para que el framework sea soportado por un sistema operativo.

-Entorno común de ejecución para lenguajes (CLR): es el núcleo del framework. El código de cualquiera de los lenguajes es compilado a un código intermedio (CIL), luego durante la ejecución del programa, a medida que se vayan invocando los eventos, este nuevo código es compilado a lenguaje máquina que se ejecuta en la plataforma del cliente.

Microsoft pretende reemplazar el API Win32 por la plataforma .NET, con el

objetivo de solventar muchos de los problemas que tiene la API, debido a que fue desarrollada sobre la marcha y sin apenas documentación.

Se ha utilizado .NET framework en este proyecto debido a que, aparte de que es la plataforma principal que compila y ejecuta código C#, proporciona un conjunto de herramientas y librerías que agilizan el desarrollo y optimizan el código como por ejemplo LINQ, que proporciona métodos para trabajar con operaciones lambda y colecciones aprovechando todos los núcleos del procesador.

1.4.5 Unity

Unity es un motor gráfico multiplataforma creado por Unity Technologies, está programado en C, C++ y C#, y se encuentra disponible para Microsoft Windows, OS X y Linux. La Figura 1.6 muestra el logotipo de Unity.

Figura 1.6: Logotipo de Unity.

Unity Technologies fue fundada en 2004 por David Helgason (CEO), Nicholas Francis (CCO), y Joachim Ante (CTO) en Copenhague, Dinamarca; con el objetivo de ser una desarrolladora de videojuegos crearon su primer videojuego GooBall, el cual fue un fracaso en ventas; sin embargo, reconocieron el potencial de sus herramientas de desarrollo y decidieron crear un motor gráfico que cualquiera pudiera usar a un precio razonable.

La primera versión de Unity apareció en la Conferencia Mundial de Desarrolladores de Apple en 2005. Originalmente fue diseñado para funcionar y generar proyectos únicamente en equipos de la plataforma Mac y obtuvo el éxito suficiente como para continuar con su desarrollo.

La Figura 1.7 muestra todas las plataformas a las que se puede exportar actualmente un proyecto de Unity.

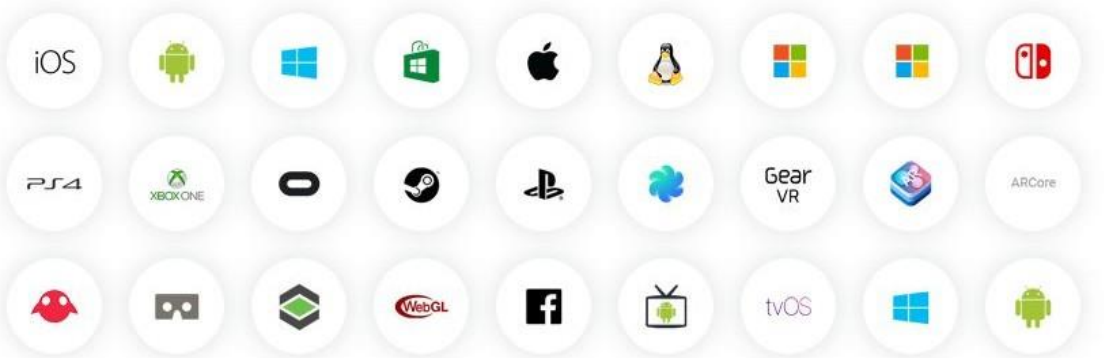

Figura 1.7: Distintas plataformas exportables por Unity.

Unity, principalmente, utiliza las siguientes tecnologías:

-Direct3D: librería gráfica que forma parte de DirectX, es propiedad de Microsoft. Es utilizada para proyectos que se ejecutan en sistemas Windows de 32 o 64 bits y la familia de consolas Xbox.

-OpenGL: librería gráfica desarrollada por Silicon Graphics. Es utilizada para proyectos que se ejecutan en sistemas donde no se encuentra DirectX, como puede ser Linux, Mac e incluso en Windows. Además, existe una versión para sistemas embebidos para Android e iOS.

-PhysX: motor físico desarrollado por AGEIA y NVIDIA. Es utilizado en todos los proyectos.

-Mono: implementación de código abierto de .NET Framework desarrollado por Xamarin. Es lo que permite que los proyectos se puedan ejecutar en casi cualquier sistema. Más adelante se detallará esta tecnología.

Originalmente Unity ofrecía a los programadores tres lenguajes de programación: C#, UnityScript (basado en javascript) y Boo (basado en Python); sin embargo, con el paso del tiempo, decidieron ofrecer soporte y compatibilidad solamente con C# debido a que la mayoría de la comunidad

trabajaba con dicho lenguaje.

Por último, mencionar que en Unity hay un orden de ejecución definido en los eventos, es decir, los scripts tienen un ciclo de vida, lo que favorece (y ofrece nativamente) la programación concurrente mediante multitareas cooperativas.

Se ha decidido desarrollar el proyecto con Unity debido a problemas de compatibilidad con algunas librerías en el uso del micrófono, ya que usaban directamente DirectX y por lo tanto no es funcional en sistema Linux, al contrario que el uso del micrófono desde Unity, que es independiente de otras tecnologías.

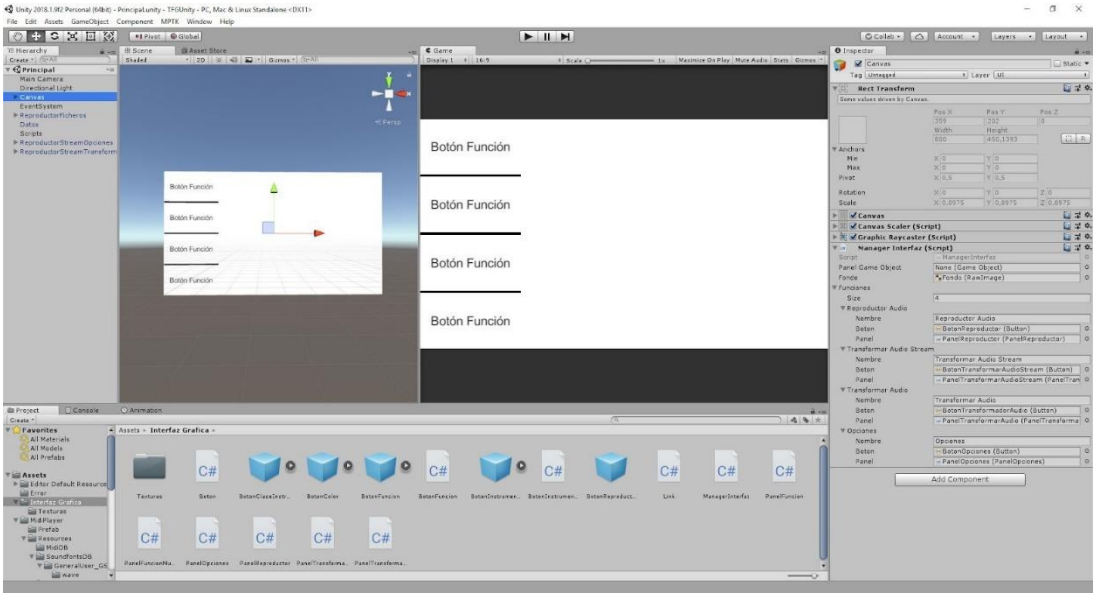

Figura 1.8: Editor de Unity.

1.4.6 Mono

Mono es un proyecto de código abierto programado en C, C# y XML, iniciado por Miguel de Icaza (cofundador de Ximian, Xamarin y GNOME) para desarrollar diversas herramientas basadas en GNU/Linux y compatible con .NET Framework, su primera versión apareció el 30 de junio de 2004. El proyecto fue creado en el año 2000 por la empresa Ximian y apoyado por Microsoft; en 2011 Ximian fue comprada por Novell, y la administración oficial del proyecto pasó a manos de Xamarin, la cual pasó a ser propiedad de Microsoft en 2016.

El proyecto Mono actualmente está compuesto principalmente por estas herramientas:

-Una máquina virtual para la infraestructura de lenguaje común (CLI).

-Una biblioteca que funciona en cualquier lenguaje compatible con CLR (Common Language Runtime).

-Compilador para C#, Visual Basic, Java y Python.

Actualmente Mono es compatible con sistemas GNU/Linux, OpenBSD, FreeBSD, UNIX, Mac OS X, Solaris y plataformas Windows.

Se ha utilizado Mono en el proyecto debido a que Unity depende de esta tecnología como ya se ha explicado anteriormente.

1.4.7 Microsoft Visual Studio

Microsoft Visual Studio es un entorno de desarrollo integrado (IDE) diseñado originalmente para sistemas operativos Windows, actualmente también es compatible con sistemas macOS y está programado en C y C#.

Visual Studio permite desarrollar aplicaciones de cualquier índole con la plataforma .NET, por lo que permite trabajar con multitud de lenguajes de programación y frameworks. Además, ofrece un repositorio online donde los usuarios pueden descargar y subir librerías y plugins para el editor, lo que reduce la complejidad de crear ciertas funcionalidades en una aplicación.

Microsoft Visual Studio apareció por primera vez en 1997 como parte de la suite de desarrollo de Microsoft, esta versión contenía herramientas de desarrollo para los lenguajes C++, Basic, InterDev, FoxPro y J++.

Se ha elegido Microsoft Visual Studio como IDE principal del proyecto debido a

que está altamente ligado con la tecnología .NET; además, es el editor que trae por defecto Unity junto con su plugin para la depuración. Concretamente se ha trabajado con la versión Community 2017 que es gratuita.

1.4.8 Geany

Geany es un editor de texto multiplataforma basado en Scintilla, con herramientas básicas para el desarrollo, está desarrollado en C y se distribuye como software libre.
Algunas de sus principales características son:

-Resaltado de sintaxis

-Autocompletado

-Soporte para plugins

-Permite compilar y ejecutar código

-Reconoce muchos tipos de archivos como, por ejemplo: C, C++, Java, PHP y Python.

Geany ha sido escogido como editor de texto para la parte del proyecto desarrollada en Linux, ya que viene con el sistema operativo y facilita el desarrollo con Python.

1.4.9 Essentia

Essentia es una librería escrita en C++ para el análisis de audio. La librería es de código abierto y contiene una versión para Python. Essentia contiene algoritmos que implementa la entrada y salida de audio, caracterización de los datos y diversos descriptores de música espectral. Ha sido creada por el Music Technology Group de la Universidad Pompeu Fabra de Barcelona.
Aunque en su repositorio de GitHub *https://github.com/MTG/essentia* afirman que es multiplataforma siendo compatible con sistemas Linux, MacOS, Windows,

iOS y Android, en su manual de instalación explican que la versión de Python no es compatible con Windows y que para se ejecute en Android, iOS y Windows la versión en C++ hay que compilarlo de forma cruzada desde Linux o Mac OS.

Se ha utilizado Essentia en este proyecto debido a que el trabajo de la Universidad de New York llamado "*A Dataset and Taxonomy for Urban Sound Research*", del cual se ha basado este trabajo, explica que emplearon esta misma librería para su estudio.

1.4.10 Accord.NET

Accord.NET es un framework para la computación cientifica para .NET, es de código abierto y está escrito completamente en C#. Fue creada por César Roberto de Souza y contiene librerías para machine learning, procesamiento de video, audio, señales, visión computacional y estadística; además, permite su uso comercial. Se puede descargar gratuitamente desde NuGet con Visual Studio o desde su repositorio de GitHub *https://github.com/accord-net/framework*. Cuenta con versiones para .NET y Mono. En su página web *http://accord-framework.net/* cuenta con una especificación de su API aceptablemente detallada.

Se ha escogido Accord.NET debido a que es el framework más completo sobre análisis de audio e inteligencia artificial que hemos encontrado para la plataforma .NET. En este proyecto se han utilizado tanto la versión para .NET como para la de Mono.

1.4.11 NAudioUnity

NAudioUnity es una adaptación de la librería NAudio para la versión de Mono que es utilizada por Unity. La adaptación fue realizada por el usuario de GitHub con alias "WulfMarius".
NAudio es una librería de audio y MIDI de código abierto para .NET, creada por Mark Heath y está escrita completamente en C#. Se puede descargar gratuitamente desde NuGet con Visual Studio o desde su repositorio de GitHub

https://github.com/WulfMarius/NAudio-Unity.

En este proyecto se ha utilizado NAudioUnity para poder leer el contenido de los archivos de audios. También es usada por la librería (explicada más adelante) encargada de leer y reproducir archivos MIDI.

1.4.12 DryWetMidi

DryWetMidi es una librería de código abierto escrita completamente en C# para trabajar con archivos y dispositivos MIDI. Fue creada por Maxim Dobroselsky y algunas de las funcionalidades que ofrece esta librería son:

-Leer, escribir y crear archivos MIDI.

-Enviar y recibir eventos MIDI de dispositivos MIDI.

-Reparar archivos MIDI corruptos.

-Reproducir archivos MIDI.

Se puede descargar gratuitamente desde NuGet con Visual Studio o desde su repositorio de GitHub *https://github.com/melanchall/drywetmidi*.

Se ha utilizado esta librería para poder crear archivos con el estándar MIDI.

1.4.13 UnitySimpleFileBrowser

UnitySimpleFileBrowser es un asset creado por Süleyman Yasir, escrito en C# y Java, está disponible de forma gratuita en la Unity Asset Store *https://assetstore.unity.com/packages/tools/gui/runtime-file-browser-113006*. Este asset contiene el código fuente de la implementación de un explorador de archivos utilizando la programación concurrente nativa de

Unity.

Ha sido necesario trabajar con este asset para poder dar al usuario una mejor experiencia y tenga una ayuda a la hora de escribir las rutas de los ficheros.
Además, este asset al estar implementado con la corrutinas de Unity no interrumpe en el ciclo de vida del motor por lo que el usuario puede estar reproduciendo un audio y con el explorador de archivos abierto al mismo tiempo y sin problemas.

1.4.14 MidiPlayerToolKitUnity

MidiPlayerToolKitUnity es un asset creado por Thierry Bachmann, escrito en C# y cuanta con dos versiones, una gratuita y otra de pago, ambas están disponibles en la Unity Asset Store *https://assetstore.unity.com /packages/tools/audio/midi-tool-kit-free-107994.* Este asset contiene la implementación de reproductor MIDI para archivos y algoritmos utilizando las librerías nativas de Unity.

En este proyecto ha sido necesario utilizar este asset para poder reproducir tanto los archivos como los eventos MIDI generados por la aplicación. Concretamente se ha trabajado con la versión gratuita del asset.

1.4.15 JSON.NET

JSON.NET es una librería de código abierto escrita en C# para trabajar con el formato JSON en .NET de forma eficiente. Fue creada por Newtonsoft y ofrece soporte para la conversión de XML a JSON y es compatible con Mono y Xamarin. Se puede descargar gratuitamente desde NuGet con Visual Studio.
Esta librería se ha utilizado para poder trabajar con JSON de forma eficiente y sencilla.

1.4.16 Metodología ágil

Las metodologías ágiles se basan en desarrollos incrementales e iterativos, al contrario de las metodologías tradicionales de desarrollo lineal o cascada. El desarrollo ágil fomenta respuestas rápidas y flexibles al cambio de los requisitos gracias a la planificación adaptativa, la identificación de requisitos colaborativos y la racionalización entre el equipo interfuncional autoorganizado.

Las principales ventajas de las metodologías ágiles son:

-Calidad: el sistema se adapta mejor a las necesidades variables del cliente, por lo que se reduce la existencia de errores en los entregables.

-Colaboración de equipo: todos los miembros del equipo están directamente involucrados en el proyecto diariamente.

-Productividad: desde etapas tempranas del desarrollo ya existe una versión ejecutable del sistema.

-Rapidez: se reduce el tiempo a la hora de implementar cambios en el sistema.

1.4.17 Programación Extrema Personal

Extreme Programming (XP) es una metodología de desarrollo de software pensada para mejorar la calidad del software y la capacidad de respuesta a los cambios en los requisitos del cliente. Es un tipo de desarrollo de software ágil por lo que alega ciclos de desarrollo cuyo fin es mejorar la productividad y la adopción de nuevos requisitos. Esta metodología fue creada en los años 90 por Kent Beck durante su trabajo en el proyecto Chrysler Compensensive Compensation System.

Los valores originales de XP son:

-Simplicidad: Se simplifica el diseño para agilizar el desarrollo y facilitar el

mantenimiento; además, el código debe comentarse en su justa medida, intentando que el código esté autodocumentado mediante la elección adecuada de los nombres de las variables, métodos y clases.

-Comunicación: Cuanto más simple sea el código mejor, ya que es más inteligible y facilita la compresión de este por otros miembros del equipo. Se recomienda la programación por parejas y es necesario que el cliente forme parte del equipo.

-Retroalimentación: Debido a que el cliente forma parte del equipo de desarrollo, su opinión sobre el estado del proyecto se conoce en tiempo real, por lo que los programadores se pueden centrar en los requisitos más importantes según el cliente.

-Coraje: Diseñar lo justo para la implementación a realizar en el presente sin pensar en implementaciones futuras. Esto evita dedicar demasiado tiempo a la parte de diseño y poder asignárselo a la parte de implementación. El código se debe reconstruir solamente si es necesario.

-Respeto: No se puede realizar cambios que hagan que las pruebas existentes fallen o dificulten, y por lo tanto demoren, el trabajo de los demás integrantes del equipo.

Los requisitos en Extreme Programming se obtienen a partir de las historias de usuario. Las historias de usuario son pequeñas descripciones del sistema en las cuales el usuario describe las funcionalidades que el software debe implementar para poder satisfacer las necesidades de dicho usuario.

Programación Extrema Personal o, en inglés Personal Extreme Programming (PXP), es una adaptación de la metodología Extreme Programming para equipos formados por un único integrante.

2

Metodología

2.1 Programación Extrema Personal

En este proyecto se ha empleado esta metodología de desarrollo. Continuando con lo explicado en el apartado de tecnologías utilizadas, vamos a ver de qué se compone el proceso iterativo de esta metodología. Dicho proceso iterativo es el mismo que hemos seguido a la hora de desarrollar nuestra aplicación.

El proceso iterativo en PXP es el siguiente:

-Requisitos: Al principio se redacta un documento con los requisitos funcionales y no funcionales del sistema

-Planificación: Se crea un conjunto de tareas y subtareas a partir de los requisitos y se estima su esfuerzo. Además, se toman las principales decisiones de diseño.

-Inicialización de la iteración: Se selecciona un conjunto de tareas las cuales serán de máxima prioridad en la iteración.

-Diseño: Se modela los elementos del sistema y sus funciones.

-Implementación: Se implementa el código, se realizan las pruebas y en el caso de que las pasen se refactoriza dicho código.

-Pruebas del sistema: Se verifica que la solución implementada cumple los requisitos del proyecto.

-Retrospectiva: Se analiza todos los datos obtenidos en la iteración. Se comprueba la exactitud en las mediciones de esfuerzo calculadas anteriormente y se analizan propuestas para mejorar el proceso.

La Figura 2.1 muestra un esquema del procedimiento descrito.

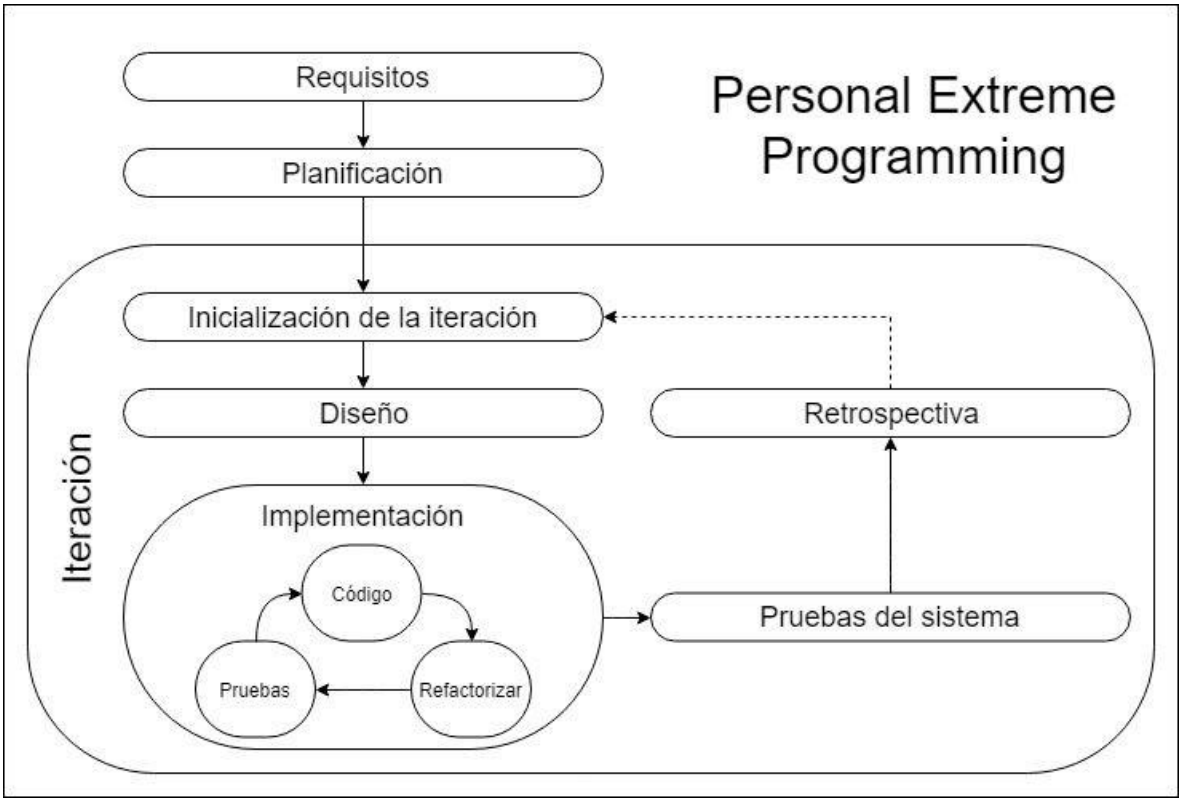

Figura 2.1: Esquema Personal Extreme Programming.

Se ha escogido esta metodología debido al principal objetivo de conseguir una

aplicación funcional dentro de un límite de tiempo, y además a la posibilidad de posibles cambios en los requisitos a causa de la dificultad del proyecto y las escasas referencias a otros proyectos similares a este, ya que la aplicación de dicha metodología es bastante adecuada para proyectos con requisitos imprecisos y cambiantes.

3

Requisitos

3.1 Análisis de los requisitos

El objetivo de esta fase es analizar las reuniones realizadas con el tutor y obtener los requisitos funcionales y no funcionales.

3.1.1 Requisitos funcionales

Los requisitos funcionales describen las características o funcionalidades que debe implementar un sistema, por lo tanto, a partir de estos se obtiene el comportamiento del sistema.

Debido a que se está empleando una metodología ágil, los requisitos funcionales han contemplado diversos cambios a lo largo del desarrollo del proyecto. A continuación, se describen los requisitos funcionales del sistema.

RF.1 El sistema debe registrar audio del entorno. El sistema tendrá que, mediante un dispositivo de entrada de audio, obtener las muestras del sonido del entorno para poder procesarlas.

RF.2 El sistema debe clasificar el audio. El sistema tendrá que clasificar las muestras del audio para poder obtener la procedencia de este, por ejemplo: perro, taladro, etcétera.

RF.3 El sistema debe identificar las frecuencias más significativas de un audio. El sistema tendrá que obtener las frecuencias existentes en el conjunto de las muestras del audio para poder seleccionar las más representativas y así conseguir las distintas notas que conforman el audio.

RF.4 El sistema debe obtener las muestras de audio de un archivo con formato WAV. El sistema tendrá que poder leer archivos WAV con el propósito de conseguir las muestras para, entre otras cosas, entrenar al clasificador.

RF.5 El sistema debe permitir escoger al usuario el número de voces, la duración de las notas y los instrumentos que desee escuchar. El sistema tendrá que poder dar la opción al usuario de cambiar los parámetros a la hora de transformar el sonido con el fin de adaptar el resultado a sus gustos.

RF.6 El sistema debe crear un archivo MIDI a partir de los datos recabados en RF.3 y RF.2. El sistema tendrá que analizar dichos datos con el fin de crear un archivo MIDI con una composición con distintos instrumentos según la clasificación del audio realizada y las notas obtenidas de las frecuencias más significativas.

RF.7 El sistema debe poder reproducir eventos MIDI. El sistema tendrá que ser capaz de ejecutar los eventos del formato MIDI.

3.1.2 Requisitos no funcionales

Los requisitos no funcionales describen las características o restricciones que no son en sí una funcionalidad, por lo tanto, a partir de estos se obtienen las propiedades del sistema, como rendimiento, seguridad, disponibilidad y portabilidad, entre otros.

Al igual que los requisitos funcionales, los no funcionales también han contemplado diversos cambios a lo largo del desarrollo del proyecto. A continuación, se describen los requisitos no funcionales del sistema.

RNF.1 La aplicación debe ser compatible con Linux y Windows.

RNF.2 La aplicación debe permitir la interoperabilidad de sus principales funcionalidades.

RNF.3 La aplicación debe utilizar el formato JSON para la serialización de datos.

RNF.4 La aplicación debe conservar la persistencia de los datos introducidos por el usuario.

RNF.5 La aplicación debe de ser eficiente a la hora de transformar el sonido para que el usuario no note retraso cuando este se reproduzca.

4

Preparación del desarrollo

4.1 Obtención de la documentación

El objetivo de esta fase es recopilar todo tipo de información teórica necesaria para poder desarrollar el proyecto. Concretamente es necesario estudiar conceptos fundamentales sobre acústica, transformada de Fourier y cómo se registra digitalmente el sonido, dándole especial interés al formato WAV.

4.1.1 Acústica

La acústica es la rama de la física que estudia las ondas mecánicas que se propagan a través de materia sólida, líquida o gaseosa; además, existe una rama en la ingeniería llamada ingeniería acústica, la cual estudia las aplicaciones tecnológicas de esta. En acústica el sonido es considerado como una vibración la cual se propaga normalmente en el aire a una velocidad de aproximadamente 343 m/s a una presión de 101325 Pa (1 atmósfera) y una temperatura de 293.15 K (20 ºC).

A continuación, vamos a ver las propiedades del sonido:

-Altura: es determinada por la frecuencia de la onda e indica si el sonido es grave o agudo, cuanto mayor sea la frecuencia, más agudo es el sonido.

-Duración: es el tiempo durante el cual se mantiene el sonido, de este se determina si el sonido es largo o corto, cuanto mayor sea la duración, más largo es el sonido.

-Intensidad: es determinada por la amplitud de la onda e indica si el sonido es fuerte o débil, cuanto mayor sea la amplitud más fuerte es el sonido.

-Timbre: es la cualidad del sonido que permite identificar su fuente, esto nos facilita diferenciar una misma nota tocada por una flauta o por una guitarra.

4.1.2 Transformada de Fourier

La transformada de Fourier, llamada así por su autor Joseph Fourier, es una transformación matemática empleada para transformar señales en función del tiempo y la frecuencia.

$$g(\xi) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{+\infty} f(x) e^{-i\xi\, x} dx$$

Figura 4.1: Transformada de Fourier.

Esta fórmula matemática descompone la señal en un conjunto de frecuencias las cuales son las que se escuchan realmente, simulando así la función del oído humano.
Veamos un caso de uso, la Figura 4.2 muestra una onda la cual corresponde al sonido producido por un sintetizador.

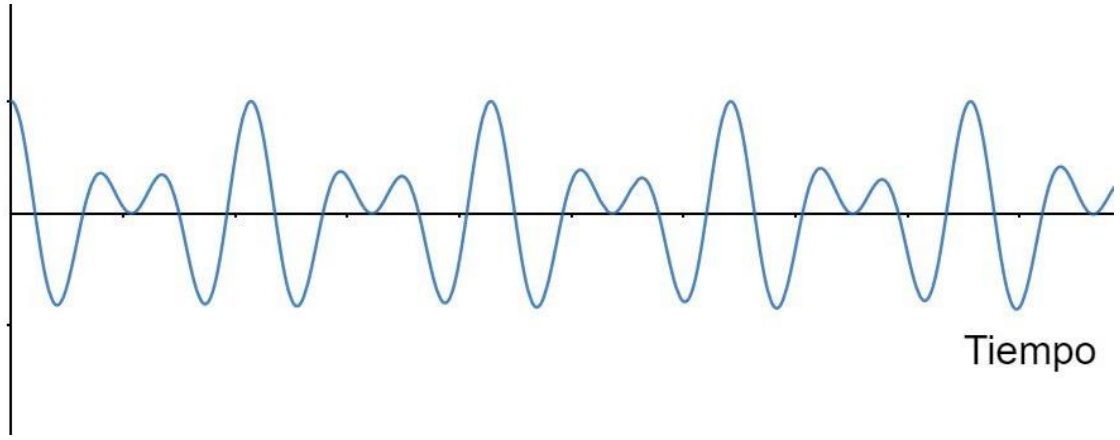

Figura 4.2: Ejemplo de onda producida por un sintetizador.

A continuación, al aplicar la transformada de Fourier a dicha señal, obtenemos las frecuencias que en su conjunto producen la onda anterior, las Figuras 4.3 y 4.4 muestran dichas frecuencias.

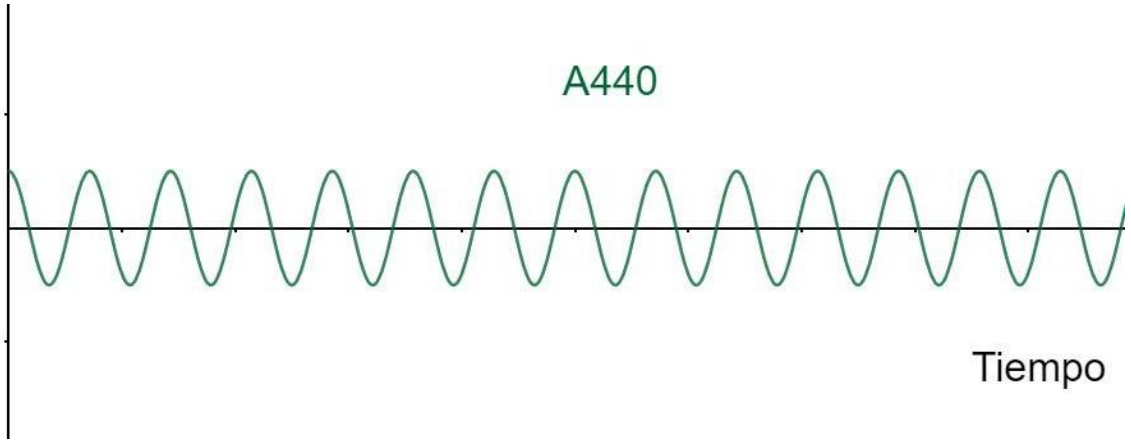

Figura 4.3: Primera frecuencia obtenida de la transformada de Fourier.

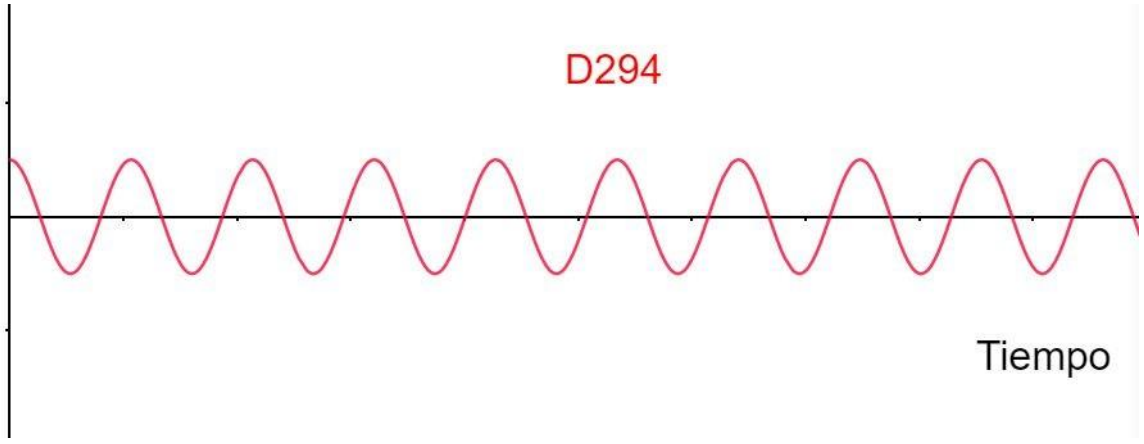

Figura 4.4: Segunda frecuencia obtenida de la transformada de Fourier.

Las tres últimas figuras se han realizado con la herramienta GeoGebra *https://www.geogebra.org/graphing?lang=es*.

4.1.3 Registro digital del sonido

Teniendo en cuenta que el sonido es cualquier fenómeno que provoca una propagación de ondas mecánicas, sean estas audibles o no; un audio digital consiste en la codificación con valores discretos de una señal eléctrica, la cual es una transformación de una onda mecánica recogida a través de un micrófono. También existe la posibilidad de registrar un conjunto de instrucciones con el fin de que, al ser interpretadas por un dispositivo compatible, se pueda reproducir determinadas frecuencias, por ejemplo, el formato MIDI.

Dicha codificación es la unión de dos procesos: el muestreo y la cuantificación digital de la señal eléctrica.

-Muestreo

El muestreo consiste en la recoger diversas tomas, en intervalos regulares, de la amplitud de la señal eléctrica; a la relación del número de muestras recogidas por tiempo se le denomina tasa de muestreo.

Según el Teorema de muestreo de Nyquist-Shannon para poder registrar un determinado intervalo de frecuencia es necesario una tasa de muestreo de poco más del doble, de este modo, si se desea cubrir el espectro audible del ser

humano (20 a 20000 Hz) se ha de emplear una tasa de muestreo superior a 40000 Hz, por lo general se suele utilizar una tasa de muestreo de 44100 Hz.

-Cuantificación digital

La cuantificación consiste en convertir las muestras obtenidas en el proceso anterior a un valor discreto de rango finito según su presión sonora. Con el objetivo de minimizar las consecuencias de la cuantificación, hay diversas técnicas según los intereses finales: lineal o uniforme, no lineal o no uniforme, logarítmica y vectorial; por ejemplo, la cuantificación logarítmica es usada en el campo de las comunicaciones telefónicas debido a que captan mejor la voz humana.

El resultado de la cuantificación depende de la cantidad de bits empleados para representar la muestra, por ejemplo, en la cuantificación lineal, si usamos 8 bits podremos distinguir 256 niveles de señales diferentes, por lo general se emplea 16 bits que ofrece una distinción de 65536 niveles de señales, lo que permite registrar sonido a 90 dB.

Una vez codificado el sonido, este se puede guardar dicha codificación en dos formatos: PCM o comprimidos; en el caso del conjunto de acciones se denominan descriptivos.

-Formatos PCM (Modulación por Impulsos Codificados)

Son los de mejor calidad, esto es debido a que almacenan toda la información procedente de la cuantificación sin ninguna omisión, por lo que también, son los formatos más pesados. Por ejemplo, WAV.

-Formatos comprimidos

Los datos obtenidos de la cuantificación son procesados por ciertos algoritmos para descartar la información que no es perceptible por el oído humano, de esta forma se puede conseguir, a cambio de sacrificar la calidad del audio, una reducción de la décima parte de lo que ocuparía un archivo con formato PCM. Por ejemplo, MP3.

-Formatos descriptivos

No almacena sonido captado por un micrófono, sino las indicaciones para que un dispositivo compatible con el formato pueda interpretar dichas órdenes y reproducir determinadas frecuencias o realizar otras acciones. Por ejemplo, MIDI.

Como se mencionó anteriormente, vamos a centrarnos en el formato WAV, debido a que es el formato no descriptivo que más influye en el proyecto.

-WAV

WAV o WAVE es un formato de audio digital desarrollado por Microsoft e IBM en 1991, su extensión es ".wav".

WAV se utiliza normalmente con el formato PCM, por lo que al no tener pérdida de calidad se suele emplear para uso profesional, sin embargo; debido a que por lo general un minuto de grabación consume unos diez megabytes de espacio, no es tan usado en Internet. Además, es compatible con gran parte de los códecs de audio.

Una de las grandes limitaciones de este formato es el tamaño máximo que puede alcanzar el audio, provocado por el número de bytes que se dedica en la cabecera a la hora de indicar la longitud del mismo. La figura 4.5 muestra un esquema del formato WAV. La cabecera del formato tiene una longitud de 44 bytes.

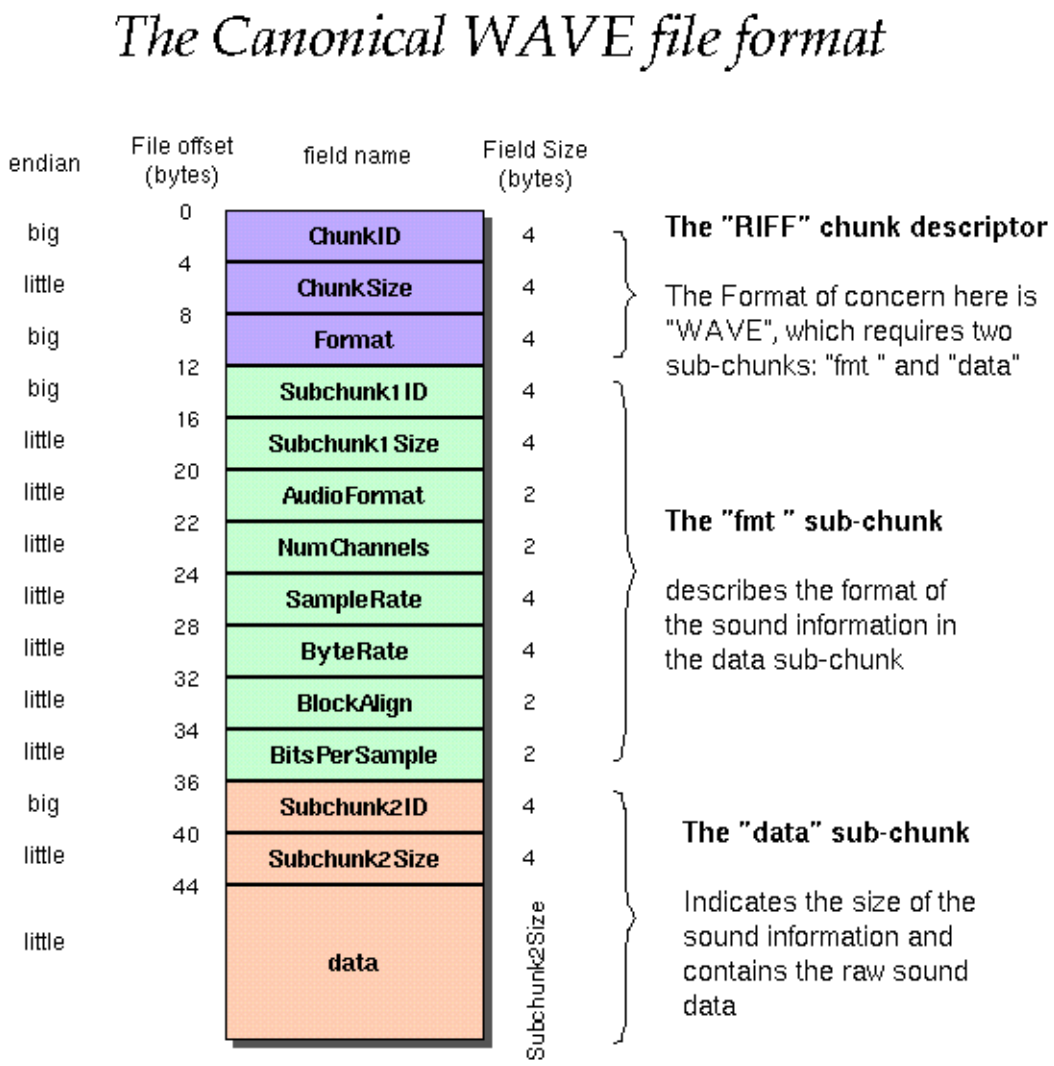

Figura 4.5: Esquema formato WAV. Soundfile.

http://soundfile.sapp.org/doc/WaveFormat/

Para terminar, vamos a comentar los campos más relevantes para nuestro proyecto:

-Subchunk1Size: indica el número de bits para la cuantificación, 16 para PCM.

-AudioFormat: indica el tipo de codificación del formato, 1 para PCM (cuantificación lineal).

-NumChannels: indica el número de canales, 1 para mono, 2 para estéreo.

-SampleRate: indica la frecuencia de muestreo, 44100 para PCM.

-BitsPerSample: indica la longitud en bits que tendrá cada muestra.

-Subchunk2Size: indica la longitud de los datos en bytes.

-Data: conjunto de muestras.

5

Desarrollo

5.1 Análisis y clasificación de un audio registrado

El objetivo de esta fase es diseñar e implementar un sistema capaz de extraer las características de las ondas procedentes de un archivo de audio para poder clasificarlo según su procedencia: animal, vehículo, etcétera.

5.1.1 Inicialización de la iteración

En esta iteración vamos a implementar dos requisitos funcionales el RF.2 El sistema debe clasificar el audio y también el RF.4 El sistema debe obtener las muestras de audio de un archivo con formato WAV. Siendo este último de mayor prioridad ya que el primero depende de este.

5.1.2 Diseño

Esta iteración va a requerir de dos proyectos. El primero consistiría en un script

de Python para poder experimentar la caracterización de los audios registrados con la librería essentia, tal cual viene explicado en el informe de la Universidad de Nueva York, llamado "*A Dataset and Taxonomy for Urban Sound Research*", en la que se basa este TFG
Este primer proyecto se encargará de realizar un análisis estadístico de las muestras que componen un audio, obteniendo de esta forma la media, varianza, covarianza, mediana, entre otras variables de dispersión.

El segundo proyecto consistirá en, primero replicar el proyecto anterior en C# utilizando Accord.NET, y segundo, entrenar un clasificador para identificar la procedencia del audio. Este proyecto estará compuesto por tres clases, la clase principal, el clasificador y una clase donde se implementarán métodos auxiliares estáticos con el fin de facilitar la lógica de la clase principal.

5.1.3 Implementación

Comenzamos con el script de Python.

El script de python está dividido en varios pasos:

1. Abrimos el fichero de audio el cual queremos caracterizar
2. Dividimos las muestras en frames y recorremos todos los frames añadiendo cada uno a un objeto "pool" propio de la librería Essentia, indicando por parámetro el nombre correspondiente a cada valor. Empleamos los mismos parámetros que se describen en el estudio de la Universidad de New York.
3. Obtenemos las distintas variables de dispersión y lo guardamos en un fichero.

A continuación, se muestra el código resultante:

```
#Abrimos el fichero de audio
ficheroSalida = open('claxon1-2.txt', 'w')
loader = essentia.standard.MonoLoader(filename='UrbanSound8K/audio/fold1/24074-1-0-7.wav')
```

```
audio = loader()

#Recorremos por frame y aplicamos la transformada de Fourier
w = Windowing(type = 'hann')
spectrum = Spectrum()
mfcc = MFCC()
pool = essentia.Pool()
for frame in FrameGenerator(audio, frameSize = 1024, hopSize = 512, startFromZero=True):
  mfcc_bands, mfcc_coeffs = mfcc(spectrum(w(frame)))
  pool.add('lowlevel.mfcc', mfcc_coeffs)
  pool.add('lowlevel.mfcc_bands', mfcc_bands)
  pool.add('lowlevel.mfcc_bands_log', logNorm(mfcc_bands))

#Obtenemos características y escribimos en el fichero
stats = ['mean', 'min', 'max', 'median', 'var', 'stdev', 'dmean', 'dvar', 'dmean2', 'dvar2', 'skew', 'kurt']
aggrPool  = PoolAggregator(defaultStats=stats)(pool)

for i in range(len(stats)):
   s = 'lowlevel.mfcc.' + stats[i]
   res = stats[i] + ': ['
   for j in range(len(aggrPool[s])):
      res += str(aggrPool[s][j]) + ', '
   res = res[:-2]
   res += ']'
   ficheroSalida.write(res + '\n')

ficheroSalida.close()
```

Continuamos con el proyecto en .NET.

Como ya se ha mencionado antes, este proyecto es una réplica del anterior con el añadido de que se implementará un clasificador el cual, es entrenado con los datos de la caracterización de diversos audios.

Para empezar, hemos ordenado el conjunto de audios descargados de la página *https://urbansounddataset.weebly.com/urbansound8k.html* según su procedencia y descartando aquellos cuyo formato no es soportado por Accord.NET, pasando de tener una colección de unos 8000 archivos a 6000. La Figura 5.1 muestra como quedan ordenados los audios.

Figura 5.1: Audios de muestra ordenados.

Una vez ordenado nuestro conjunto de audio, podemos empezar a entrenar a nuestro clasificador. Como ya se ha mencionado anteriormente, este sistema está compuesto por tres clases, vamos a ver cada una de ella.

-Utilidades

Es una clase que contiene sobre todo métodos estáticos que facilitan el ordenado de los audios, mostrar mensajes por la consola, buscar ficheros, etcétera. La Figura 5.2 muestra el diagrama de esta clase.

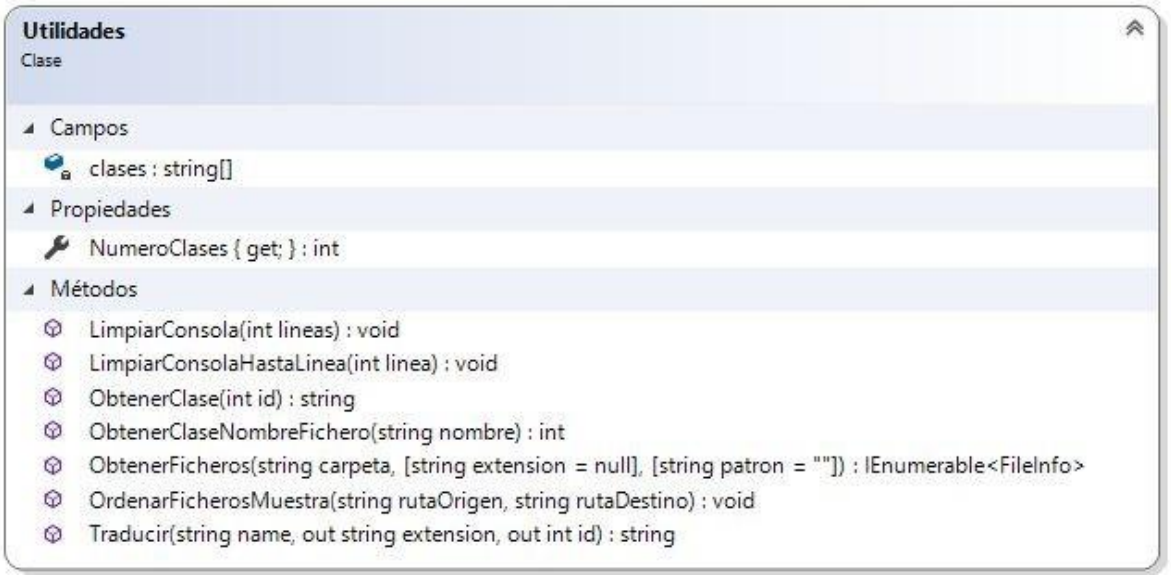

Figura 5.2: Diagrama de clase Utilidades.

Estos son algunos de los métodos implementados:

```
//Ordena los audios hijos de rutaOrigen según su fuente ignorando los que no son compatibles, el resultado se almacena en rutaDestino
public static void OrdenarFicherosMuestra(string rutaOrigen, string rutaDestino)
{
  int[] idFicheros = new int[clases.Length];
  var ficheros = ObtenerFicheros(rutaOrigen, "wav");
  var clasificador = new Clasificador("");

  var directorio = new DirectoryInfo(rutaDestino);
  foreach (var c in clases)
  {
    directorio.CreateSubdirectory(c);
  }

  int fallos = 0;
  for (int i = 0; i < ficheros.Count(); i++)
  {
    var f = ficheros.ElementAt(i);
    try
    {
      int id; string extension;
      var data = clasificador.ConseguirCaracteristicas(f.FullName);
      string nombre = Traducir(f.Name, out extension, out id);
      string ruta = rutaDestino + "/" + clases[id] + "/" + nombre + "-" + idFicheros[id] + extension;
      idFicheros[id]++;
      f.CopyTo(ruta, true);
    }
    catch
    {
      fallos++;
    }
    finally
    {
      Console.WriteLine("Ordenando archivos de audio ({0} de {1})", i + 1, ficheros.Count());
      Console.SetCursorPosition(0, Console.CursorTop - 1);
    }

  }

  Console.WriteLine("{0} archivos de audio ordenados, {1} con fallos", ficheros.Count(), fallos);
}

//Obtiene todos los ficheros hijos de una carpeta, que contenga en su nombre una extensión y un patrón
public static IEnumerable<FileInfo> ObtenerFicheros(string carpeta, string extension = null, string patron = "")
{
  var directorio = new DirectoryInfo(carpeta);
  var res = from d in directorio.GetDirectories()
            from f in d.GetFiles("*." + extension ?? "*")
            where f.Name.Contains(patron)
            orderby f.Name ascending
            select f;
```

```
  return res;
}
```

-Clasificador

Esta es la clase más importante de las tres, es la encargada de caracterizar un audio y obtener su procedencia, tiene un método para guardar el entrenamiento realizado en archivo binario y otro para cargarlo. La Figura 5.3 muestra el diagrama de esta clase.

Tras comparar los distintos algoritmos de aprendizaje que ofrece la librería Accord.NET, nos quedamos con el algoritmo KNN (K-Nearest Neighbours) ya que es que mejor resultado nos ha dado y se entrena relativamente rápido.

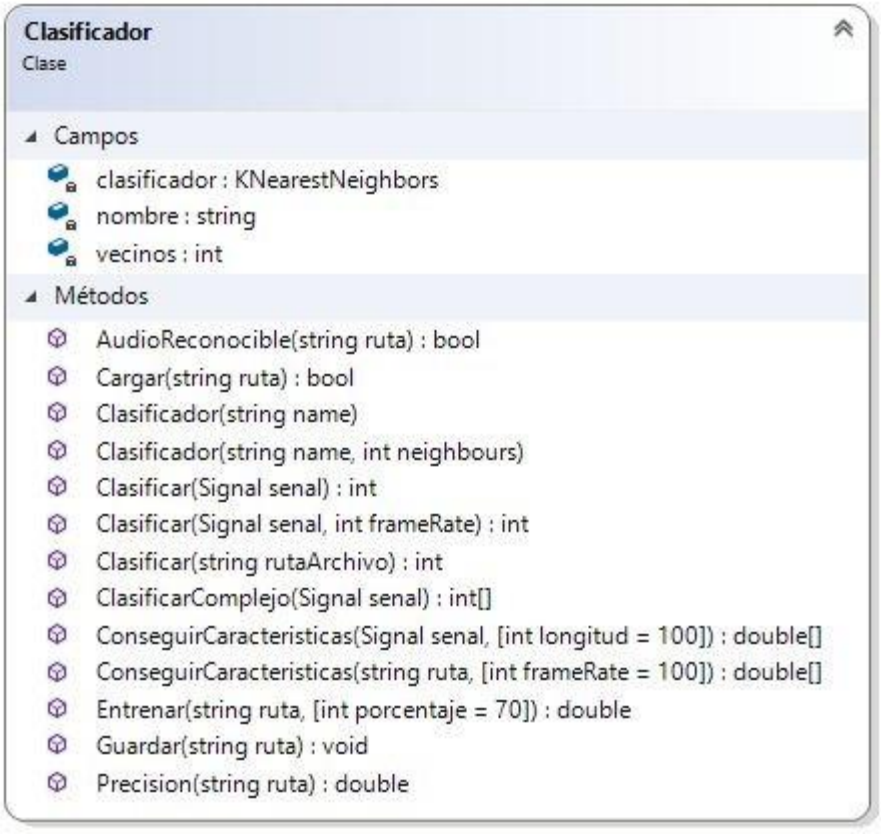

Figura 5.3: Diagrama de clase Clasificador.

A continuación, se muestra la implementación de los dos métodos más relevantes.

```
//Entrena el clasificador y devuelve la precisión del clasificador entrenado
public double Entrenar(string ruta, int porcentaje = 70)
{
  var directorio = new DirectoryInfo(ruta);
  var ficheros = Utilidades.ObtenerFicheros(ruta, "wav");
  var inputs = new List<double[]>();
  List<int> outputs = new List<int>();
  Random rnd = new Random();
  ficheros = ficheros.OrderBy(x => rnd.Next());
  int numFicheros = (int)(ficheros.Count() * porcentaje / 100f);

  int cont = 0;
  foreach (var f in ficheros)
  {
    if (cont >= numFicheros) break;
    try
    {
      inputs.Add(ConseguirCaracteristicas(f.FullName));
      outputs.Add(Utilidades.ObtenerClaseNombreFichero(f.Name));
      cont++;
    }
    catch (Exception e)
    {
      //Console.WriteLine($"Error con fichero: {f.Name}, {e.Message}");
    }
  }

  //Creamos el clasificador
  clasificador = new KNearestNeighbors()
  {
    K = vecinos, // basar una decisión en las etiquetas de clase de los vecinos más cercanos del punto
de consulta
    Distance = new Manhattan() // cálculo de la distancia de los vecinos
  };

  // Entrenamos al clasificador
  clasificador.Learn(inputs.ToArray(), outputs.ToArray());

  // Obtenemos las prediciones de las entradas que le hemos dado
  int[] predicciones = clasificador.Decide(inputs.ToArray());

  // Crea una matriz de confusión para verificar la calidad de las predicciones
  var cm = new GeneralConfusionMatrix(predicted: predicciones, expected: outputs.ToArray());

  // Compruebe la medida de precisión:
  return cm.Accuracy;
}

//Obtiene las variables estadísticas de una señal con una longitud máxima
public double[] ConseguirCaracteristicas(Signal senal, int longitud = 100)
{
  //Calculamos los coeficientes mediante la Transformada de Fourier
  var mfcc = new MelFrequencyCepstrumCoefficient(frameRate: 100);
  var coeficientes = mfcc.Transform(senal);
```

```
  //Calculamos las estadísticas
  var max = coeficientes.Select(x => x.Descriptor.Max()).ToArray();
  Array.Resize(ref max, longitud);

  var min = coeficientes.Select(x => x.Descriptor.Min()).ToArray();
  Array.Resize(ref min, longitud);

  var mediana = coeficientes.Select(x => x.Descriptor.Median()).ToArray();
  Array.Resize(ref mediana, longitud);

  var media = coeficientes.Select(x => x.Descriptor.Mean()).ToArray();
  Array.Resize(ref media, longitud);

  var varianza = coeficientes.Select(x => x.Descriptor.Variance()).ToArray();
  Array.Resize(ref varianza, longitud);

  var sesgo = coeficientes.Select(x => x.Descriptor.Skewness()).ToArray();
  Array.Resize(ref sesgo, longitud);

  var curtosis = coeficientes.Select(x => x.Descriptor.Kurtosis()).ToArray();
  Array.Resize(ref curtosis, longitud);

  //Devolvemos la unión de las estadísticas anteriormente calculadas
  var res =
max.Concat(min).Concat(mediana).Concat(media).Concat(varianza).Concat(sesgo).Concat(curtosis).T
oArray();

  return res;
}
```

-Program

Es la clase principal, contiene un método estático, el cual es llamado por el método Main() al inicio de la ejecución, este método crea una instancia del clasificador, y lo entrena en el caso de que no pueda cargarlo, posteriormente hace una prueba de precisión.

La implementación de dicho método es la siguiente:

```
static void EntrenarClasificador()
{
  string nombre = "clasificador";
  var clas = new Clasificador(nombre);
  double precision;
  if (!clas.Cargar(Directory.GetCurrentDirectory()))
  {
    Console.WriteLine("Entrenando Clasificador");
    precision = clas.Entrenar(@"C:\Users\josag\Documents\Universidad\Trabajo Fin Grado\Sonidos");
    Console.WriteLine($"Precisión audios entrenados: {precision * 100:F2}%");
    clas.Guardar(Directory.GetCurrentDirectory());
  }
```

```
    precision = clas.Precision(@"C:\Users\josag\Documents\Universidad\Trabajo Fin Grado\Sonidos");
    Console.WriteLine($"Precisión audios total: {precision * 100:F2}%");
}
```

5.1.4 Pruebas del sistema

El sistema funciona correctamente, es capaz de obtener la fuente de un audio con cierta probabilidad. A la hora de medir la precisión obtenemos un conjunto formado por audios que formaron parte del entrenamiento y los que no, comparamos si la clase que predice es correcta, y obtenemos una precisión de acierto del 84%, un porcentaje aceptable.

5.1.5 Retrospectiva

A partir de lo conseguido en el script de python y conseguir trasladarlo a .NET, ya tenemos un clasificador de archivos de audio. Esta iteración ha llevado bastante tiempo debido a fue necesario documentarse sobre cómo funcionaba el tratamiento con los archivos WAV en ambas librerías y los distintos clasificadores.

5.2 Clasificar un audio en directo

El objetivo de esta fase es conseguir, a partir del sistema anterior, una aplicación que pueda analizar y clasificar sonidos captados por el micrófono en tiempo real.

5.2.1 Inicialización de la iteración

En esta iteración vamos a implementar el requisito funcional RF.1 El sistema debe registrar audio del entorno.

5.2.2 Diseño

Se ampliará la funcionalidad de la clase principal, creando un método para obtener el sonido del micrófono, clasificar las muestras del audio y mostrar por pantalla su procedencia.

5.2.3 Implementación

Para la grabación del audio se ha decidido crear un fichero donde a medida que el micrófono capte el audio, se irá almacenando la información en dicho fichero, y luego de forma paralela, un bucle estará leyendo los datos de dicho archivo y clasificándolos. El método estático, perteneciente a la clase principal, que realiza esta funcionalidad quedaría de la siguiente forma:

```
static WaveEncoder grabadora;
static Clasificador clasificador;
static void GrabarAudio()
{
  Console.WriteLine("Grabar Audio");

  //Obtenemos el clasificador entrenado
  string nombre = "clasificador";
  clasificador = new Clasificador(nombre);
  if (!clasificador.Cargar(Directory.GetCurrentDirectory()))
  {
    Console.WriteLine("Entrenando Clasificador");
    var precision = clasificador.Entrenar(@"C:\Users\josag\Documents\Universidad\Trabajo Fin Grado\Sonidos");
    Console.WriteLine($"Precisión audios entrenados: {precision * 100:F2}%");
    clasificador.Guardar(Directory.GetCurrentDirectory());
  }

  //Creamos el lector y escritor del fichero de audio donde meteremos lo que capte el micrófono
  var ficheroEncode = new FileStream(Directory.GetCurrentDirectory() + "/audio.wav", FileMode.Create, FileAccess.Write, FileShare.ReadWrite);
  var ficheroDecode = new FileStream(Directory.GetCurrentDirectory() + "/audio.wav", FileMode.Open, FileAccess.Read, FileShare.ReadWrite);
  grabadora = new WaveEncoder(ficheroEncode);
  WaveDecoder sourceDecoder = new WaveDecoder();

  //Creamos el micrófono y empezamos a grabar
  var micro = new AudioCaptureDevice()
  {
    DesiredFrameSize = 44100,
    SampleRate = 44100
  };
  micro.NewFrame += GrabarFrame;
  micro.Start();

  Thread.Sleep(2000);

  int muestra = 0;
  int index = 0;

  while (true)
  {
    //Leemos lo que haya en el audio
    ficheroDecode.Seek(0, SeekOrigin.Begin);
    sourceDecoder.Open(ficheroDecode);
```

```
        int frames = sourceDecoder.Frames;

        //Clasificamos y emprimimos por pantalla el resultado
        var senal = sourceDecoder.Decode(index, frames - index);
        var clase = clasificador.Clasificar(senal);

        Console.WriteLine($"Muestra {muestra}: {Utilidades.ObtenerClase(clase)} ({clase})");
        muestra++;
        index = frames;

        Thread.Sleep(2000);
    }
}

public static void GrabarFrame(object sender, NewFrameEventArgs eventArgs)
{
    //Almacenamos en el fichero lo que ha captado el micrófono
    grabadora.Encode(eventArgs.Signal);
}
```

La espera de dos segundos (Thread.Sleep(2000)) es para conseguir que los fragmentos de audios a analizar tengan una duración dos segundos, tal y como la gran parte de los audios que formaron parte del entrenamiento.

Sin embargo, hay un problema, analizando la librería nos damos cuenta que la clase AudioCaptureDevice (el micrófono) internamente se hace llamadas a un DLL llamado SharpDX; si buscamos información sobre dicho compilado nos damos cuenta que es una librería para trabajar con DirectX. Esto nos podría causar un conflicto con el requisito no funcional RNF.1 La aplicación debe ser compatible con Linux y Windows, ya que DirectX es una librería gráfica para sistemas Windows no para Linux, y efectivamente, probamos nuestro proyecto de
.NET en Linux a través de Mono y llegado al punto de empezar a grabar se nos muestra un error, no encuentra dicho DLL.

Debido a esto hay que hacer un cambio de planes, tras barajar diversas soluciones, se elige finalmente trabajar con Unity, por lo que creamos un proyecto en Unity y recreamos la implementación anterior.

Finalmente, tras pasar el proyecto a Unity, y utilizando sus librerías nativas de

grabación de audio, hemos prescindido de los ficheros y directamente grabamos, esperamos y clasificamos, lo que nos quedaría de la siguiente manera:

```
[SerializeField]
private Text clasificadortxt;

IEnumerator Start()
{
  //Cargamos clasificador
  var clasificador = new Clasificador("clasificador");
  clasificador.Cargar("Clasificador");

  while (true)
  {
    //Empezamos a grabar audio con frecuencia 41000 Hz y duración 2 segundos
    var audio = Microphone.Start(null, false, 2, 41000);

    //Esperamos a que acabe de grabar
    yield return new WaitWhile(() => Microphone.IsRecording(null));

    //Obtenermos las muestras y la clasificamos
    var datos = new float[audio.samples];
    audio.GetData(datos, 0);
    int clase = clasificador.Clasificar(datos, audio.frequency);

    //Mostramos el resultado
    clasificadortxt.text = Datos.ObtenerClase(clase);
  }
}
```

La clase Datos es donde se guardará toda la información necesaria para el sistema en su conjunto, por ejemplo, el conjunto de fuentes de audio disponibles, imágenes para la interfaz de usuario, entre otros.

5.2.4 Pruebas

El sistema funciona correctamente, aunque no es tan preciso como en la anterior iteración, ya que dependiendo del micrófono y el ruido ambiente puede perjudicar la calidad del audio y por lo tanto su clasificación.

5.2.5 Retrospectiva

Gracias a pasar el proyecto a Unity ya disponemos de un clasificador de audio en directo multiplataforma. Aunque este traspaso de tecnología no estaba

previsto, los conocimientos previos en Unity han ayudado a que no se largara tanto la iteración.

5.3 Análisis musical del sonido ambiente

El objetivo de esta fase es modificar la aplicación con el fin de procesar las características de las ondas y obtener las frecuencias más significativas.

5.3.1 Inicialización de la iteración

En esta iteración vamos a implementar el requisito funcional RF.3 El sistema debe identificar las frecuencias más significativas de un audio.

5.3.2 Diseño

Actualmente tenemos tres clases: la clase principal, la clase del clasificador y la de utilidades. Para esta iteración vamos a crear dos clases más, Afinador y NotaMusical.

Afinador será la encargada de recibir un conjunto de muestras y obtener sus frecuencias más significativas, dispondrá de varios métodos con distintos parámetros para facilitar su uso tanto a la hora de analizar muestras de ficheros de audio como para las del micrófono, además de su legibilidad.

NotaMusical es la clase encargada de representar las notas junto con su octava, en ella se almacenará la posición que tendría la nota en un piano estándar de 88 teclas (siendo la tecla 49 la correspondiente a A440), su octava, su posición en un array de notas y su frecuencia; contará con diversas funciones para a partir de un subconjunto de datos obtener los demás, por ejemplo: a partir de la nota y su octava obtener la frecuencia y su posición en el piano.

5.3.3 Implementación

Comencemos por la implementación de la clase NotaMusical, ya que de esta dependerá los métodos de la clase afinador Afinador.

Para empezar, vamos a ver y explicar sus variables.

```
public enum Notas { C, CSharp, D, DSharp, E, F, FSharp, G, GSharp, A, ASharp, B }
static string[] notas = { "C", "C#", "D", "D#", "E", "F", "F#", "G", "G#", "A", "A#", "B" };

static float frecuenciaBase = 440;
static int notaBase = 9;
static int octavaBase = 4;

static int notaPrimera = 9;
static int octavaPrimera = 0;

static double razonSemitono = Math.Pow(2, 1 / 12f);

public int Nota { get; private set; }
public int Octava { get; private set; }

public float Frecuencia { get; private set; }
public int Posicion { get; private set; }
```

Antes de nada, las notas se manejan en notación anglosajona, ya que es la notación más popular.

El primer array sirve para facilitar la creación de esta clase en una clase externa, ya que es más cómodo reconocer un enumerable que un número.

El segundo array es para el método ToString(), el cual se usa para facilitar la depuración y las pruebas del sistema. Este array como en el anterior están escritos en notación anglosajona.

Las variables frecuenciaBase, notaBase, octavaBase hacen referencia a que los cálculos se efectuarán para conseguir notas afinadas en A440, ya que la afinación es un acuerdo entre músicos, y aunque el estándar es afinar en A440, hay partidarios de la afinación en A332. La Figura 5.4 muestra la nota A440 en un pentagrama.

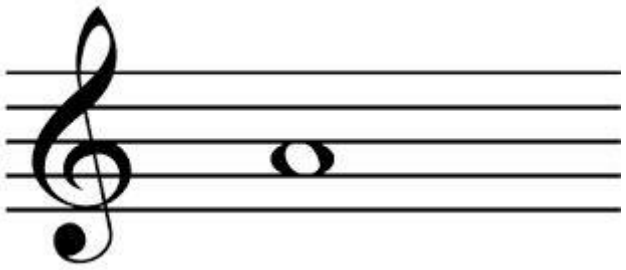

Figura 5.4: La 440 (A4, A440, La4).

La dos siguientes variables, notaPrimera y octavaPrimera indica que la primera tecla del piano (y por tanto la primera posición) corresponde a un A0 (notas[notaPrimera] + octavaPrimera), con una frecuencia de 27.5 Hz.

El valor de razonSemitono es el factor de multiplicación para subir un semitono a una frecuencia dada. Es decir que si yo tengo una frecuencia de 440 (A4) y lo multiplico por dicho factor (440 * razonSemitono) me devuelve 466,164 (A#4). Dicho factor se obtiene a partir del siguiente razonamiento.

-En una octava justa hay 5 tonos y 2 semitonos, teniendo en cuenta que 1 tono son 2 semitonos, obtenemos un total de 12 semitonos por octava.

-Por lo tanto, si tenemos una frecuencia, por ejemplo 440, y la correspondiente a su octava, 880 (el doble), obtenemos la siguiente fórmula: 440 * r^12 = 880 (siendo r la constante razonSemitono).

-Despejamos y obtenemos que r es igual a la raíz doceava de dos.

Finalmente, las cuatro variables restantes son las que se explican en el apartado de diseño.

Para acabar con esta clase vamos a ver los métodos que calculan el conjunto de propiedades a partir de un subconjunto de estos.

```
private void CrearNotaMusical(float frecuencia)
{
  if (frecuencia >= 1)
```

```
  {
    int notasHastaBase = (int)Math.Round(Math.Log((frecuencia / frecuenciaBase), razonSemitono));
    int posicion = octavaBase * notas.Length + notaBase + notasHastaBase;
    posicion -= notaPrimera * (octavaPrimera + 1) - 1;
    CrearNotaMusical(posicion);
  }
  else
  {
    CrearNotaMusical(int.MinValue);
  }
}

private void CrearNotaMusical(int posicion)
{
  if (posicion != int.MinValue)
  {
    posicion += notaPrimera * (octavaPrimera + 1) - 1;

    if (posicion < 0) posicion -= notas.Length;

    int octava = posicion / notas.Length;
    int nota = posicion - notas.Length * octava;

    if (nota < 0) nota += notas.Length;
    CrearNotaMusical(nota, octava);
  }
  else
  {
    Posicion = posicion;
    Frecuencia = -1;
  }
}

private void CrearNotaMusical(int nota, int octava)
{
  Nota = nota;
  Octava = octava;

  int notasHastaBase = (octava * notas.Length + nota) - (octavaBase * notas.Length + notaBase);
  Frecuencia = frecuenciaBase * (float)Math.Pow(razonSemitono, notasHastaBase);

  Posicion = (octava - octavaPrimera) * notas.Length + nota - notaPrimera + 1;
}
```

Por lo tanto, añadiendo unos métodos para facilitar el manejo de esta clase, la Figura 5.5 nos muestra cómo nos quedaría.

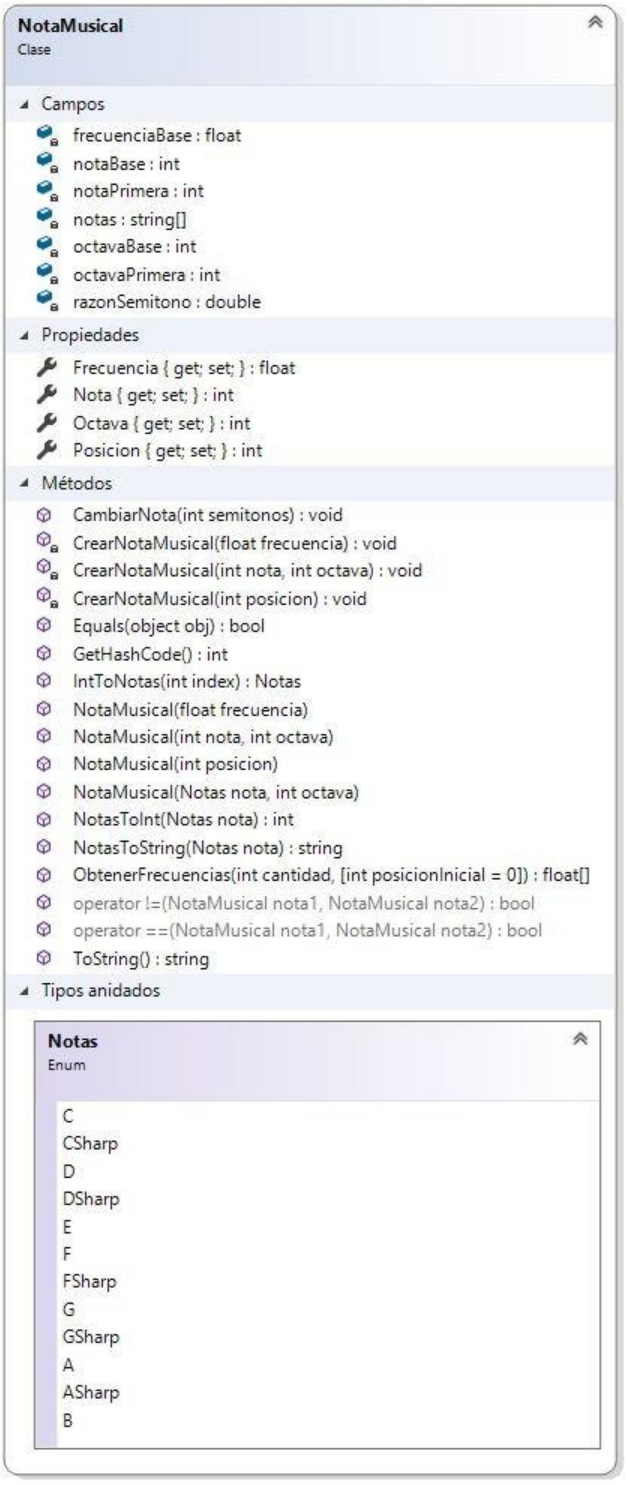

Figura 5.5 Diagrama de clase NotaMusical.

La clase Afinador contiene los métodos necesarios para obtener las frecuencias, y por lo tanto sus respectivas notas, más significativas en un conjunto de muestras, tal y como se ha explicado en el apartado de diseño.

En este caso hay un problema, y es que el silencio absoluto no existe, por lo que,

mientras que nosotros percibimos silencio, en verdad hay ruido solo que no lo captamos, pero un micrófono sí; esto es debido a que el propio cuerpo humano emite ondas mecánicas. Además, hay que tener en cuenta las ondas mecánicas producidas por el movimiento del disco duro, los ventiladores del ordenador, entre otras cosas.

Vamos a utilizar un Analizador de FFT que viene con la librería Accord.NET para ver un ejemplo.

La Figura 5.6 muestra una captura de pantalla cuando se supone que hay “silencio”, y la Figura 5.7 muestra otra captura, pero esta vez hay reproduciéndose una onda senoidal con una frecuencia de 440 Hz (A4).

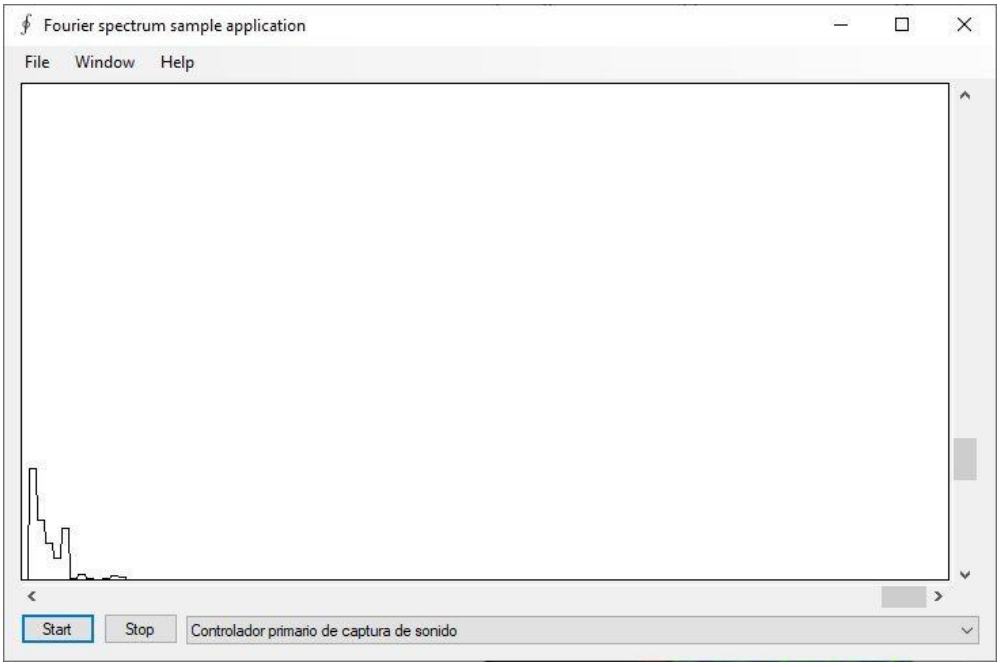

Figura 5.6: Espectro del silencio.

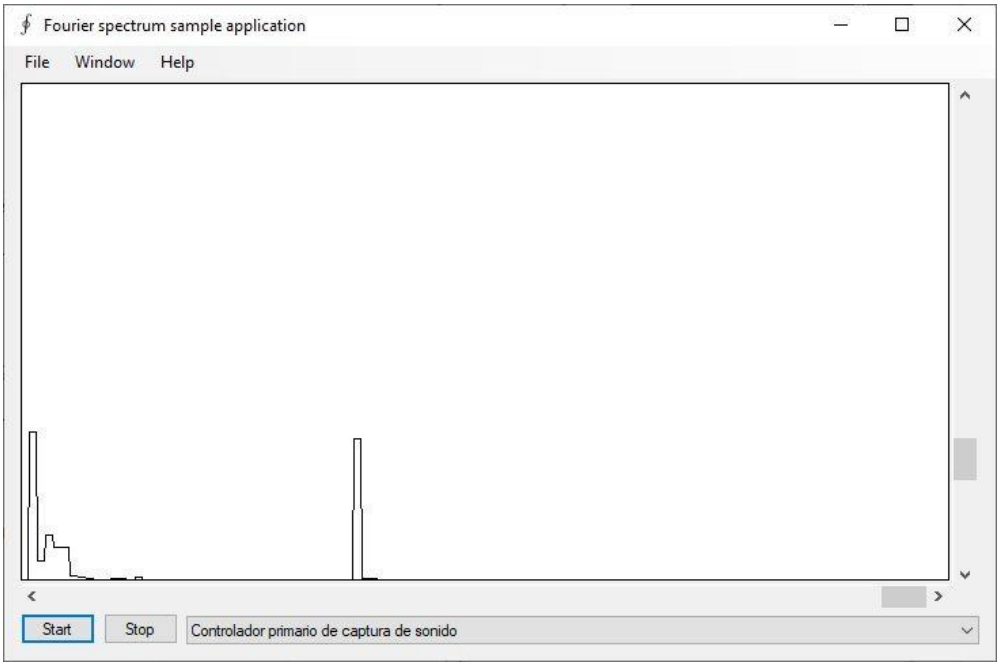

Figura 5.7: Espectro de onda senoidal a 440Hz.

Como podemos ver, en ambas se presenta una captación de ondas con bajas frecuencias (parte izquierda de la imagen) con bastante intensidad, por lo que ese ruido de fondo siempre va a estar haya “silencio” o no. Además, tenemos el problema de los armónicos, el cual se muestra en la Figura 5.8.

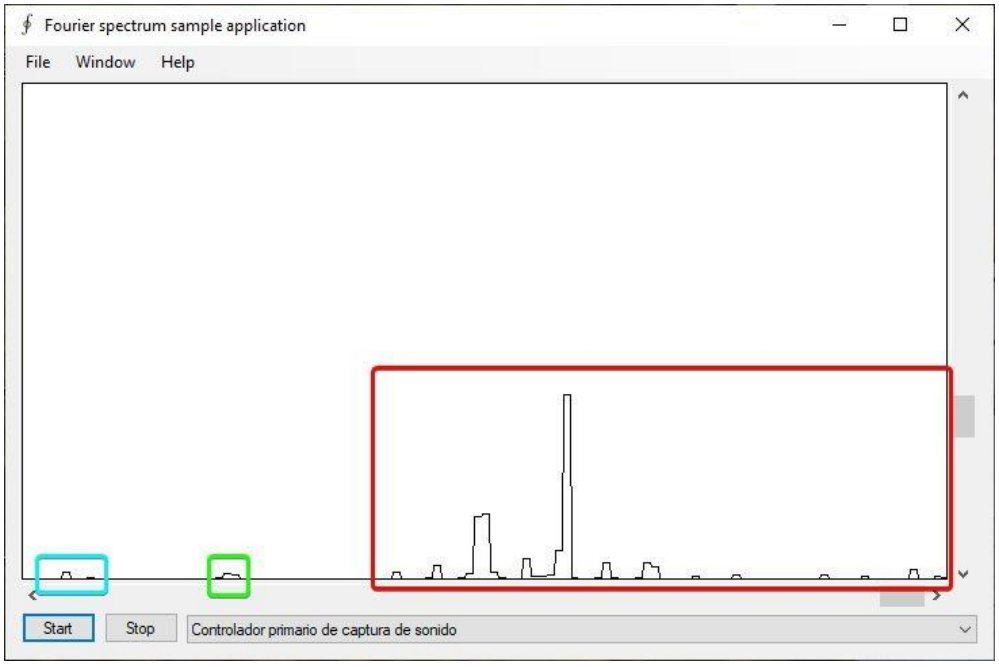

Figura 5.8: Espectro de la nota A1 tocada en un piano.

Como se puede observar, hay tres zonas marcadas por colores, el color azul marca las frecuencias correspondientes al “silencio” vista anteriormente, la zona verde marca la frecuencia base de la nota A1 (55Hz) y la zona roja marca los armónicos de dicha nota, estos armónicos son frecuencias múltiplo de la frecuencia base (55 * 4 Hz, 55 * 6 Hz, etcétera). Este conjunto de armónicos es el que define el timbre del sonido.

Buscando una solución a este problema nos encontramos con el algoritmo Goertzel. El algoritmo Goertzel es una técnica en el procesamiento de señales digitales, el cual es usado en sistemas empotrados debido a su simplicidad y eficiencia respecto a la transformada de Fourier, te permite detectar frecuencias específicas en una señal. Se probaron tres implementaciones distintas de este algoritmo; sin embargo, las tres tenían el mismo problema. El problema es que este algoritmo está diseñado para buscar una frecuencia específica, y nosotros necesitamos buscar varias, por lo que a la hora de probarlo no funcionaba correctamente, no era capaz de reconocer varias frecuencias en un conjunto de muestras, por lo que descartamos este algoritmo y volvemos a la transformada de Fourier.

Para solventar los problemas antes mencionados a la hora de obtener las frecuencias más significativas, vamos a seguir el siguiente proceso:

-Ponemos una frecuencia mínima y una máxima, de esta forma ignoramos las frecuencias que siempre están y los armónicos más agudos.

-Cogemos las frecuencias que tengan más potencia.

-Y ahora vemos si la potencia de cada frecuencia supera un límite, si lo supera es una nota, en caso contrario es un silencio; el límite estará relacionado con las potencias del conjunto de frecuencias y el número de notas que queramos obtener.

Por lo tanto, la implementación de este algoritmo nos quedaría de la siguiente

manera.

```
public NotaMusical[] ObtenerNotasFFT(float[] samples, int cantidad = 1, int sampleRate = 44100)
{
  var signal = Signal.FromArray(samples, sampleRate);
  return ObtenerNotasFFT(signal, cantidad);
}

public NotaMusical[] ObtenerNotasFFT(Signal senal, int cantidad = 1)
{
  if (cantidad < 1) cantidad = 1;
  var notaMinima = new NotaMusical(NotaMusical.Notas.C, 3);
  var notaMaxima = new NotaMusical(NotaMusical.Notas.C, 7);
  ComplexSignal signal = ComplexSignal.FromSignal(senal);
  signal.ForwardFourierTransform();
  double[] power = Tools.GetPowerSpectrum(signal.GetChannel(0));
  double[] freqv = Tools.GetFrequencyVector(signal.Length, signal.SampleRate);
  double max, min;
  ObtenerMaxMin(power, out max, out min);
  double potenciaMinima = (max - min) / (cantidad + 1) + min;
  return power.Select((p, i) => new { p, i }).Where(x => freqv[x.i] >= notaMinima.Frecuencia &&
freqv[x.i] <= notaMaxima.Frecuencia)
    .OrderByDescending(x => x.p).Take(cantidad)
    .Select(x => x.p > potenciaMinima ? new NotaMusical((float)freqv[x.i]) : null).ToArray();
}

private void ObtenerMaxMin(IEnumerable<double> datos, out double max, out double min)
{
  max = double.MinValue;
  min = double.MaxValue;

  foreach(var dato in datos)
  {
    if (dato > max) max = dato;
    if (dato < min) min = dato;
  }
}
```

5.3.4 Pruebas

El sistema es capaz de reconocer las notas tocadas por un piano y una guitarra, siempre y cuando estén dentro del rango de frecuencia deseado. Sin embargo, para ciertas notas, sobre todos las más graves, tienen una potencia menor que sus armónicos y por lo tanto son ignoradas por el sistema, por ejemplo: se reproduce un G3 (196Hz) y el sistema capta un G4 (392 Hz). Este problema también aparece en los afinadores comerciales, por lo que, aunque hemos reducido su efecto, estará presente.

5.3.5 Retrospectiva

Aunque para esta iteración se haya tenido que implementar y probar múltiples algoritmos, el tiempo dedicado no ha excedido al planeado.

5.4 Transformación del sonido

El objetivo de esta fase es poder utilizar la información obtenida en el análisis musical y en la clasificación del sonido, con el fin de obtener las notas más adecuadas y sus timbres conforme a unas determinadas reglas.

5.4.1 Inicialización de la iteración

En esta iteración vamos a comenzar la implementación del requisito RF.6 El sistema debe crear un archivo MIDI a partir de los datos recabados en RF.3 y RF.2.

5.4.2 Diseño

Para alcanzar los objetivos de esta iteración, se van a crear tres clases: FiguraMusical, Composicion y Compositor.

FiguraMusical es la clase encargada de representar la duración de una NotaMusical. Tiene tres atributos, una referencia a la NotaMusical que representa (si es nulo representa un silencio), un booleano que indica si es silencio o no y un número entero que indica la duración de la nota en milisegundos. Contiene además varios constructores para crear a la vez la nota y la figura, dada una frecuencia, posición o nota con su octava.

Composicion es la clase que representa una partitura, contiene tres variables, una referencia al fichero MIDI en la cual se almacenarán los datos, una referencia al tempo (de tipo Tempo) y un array de voces (de tipo PatternBuilder). Dispondrá de métodos para añadir notas a las distintas voces, añadir acordes, cambiar los instrumentos de las distintas voces y guardar los datos en un fichero MIDI. Además, las notas se añadirán a su respectiva voz dependiendo de la intensidad con la que ha sido captada su frecuencia, de esta manera las notas con más

intensidad se escucharán más fuertes.

Compositor es la clase más importante de la iteración, es la clase encargada de manejar el clasificador y el afinador para obtener los datos de un conjunto de muestras y añadir las notas y los instrumentos a la Composicion. Es decir, esta clase dispondrá de métodos los cuales recibirán un conjunto de muestras o bien un fichero de audio y a partir del uso del clasificador y del afinador creará una composición. Ya que se va a trabajar directamente con esta clase y no con Composicion, se ha decidido que herede de esta última.

5.4.3 Implementación

Debido a que Compositor y Composicion dependen de FiguraMusical, esta será la primera a implementar.
Como ya se ha explicado en el apartado anterior, estas serían sus variables.

```
private NotaMusical nota;
private bool silencio;

public int Duracion { get; set; } //Milisegundos

public bool Silencio
{
  get { return silencio; }
  set
  {
    silencio = value;
    if (silencio) nota = null;
  }
}

public NotaMusical Nota
{
  get { return nota; }
  set
  {
    nota = value;
    silencio = value == null;
  }
}
```

Y estos son los distintos constructores mencionados anteriormente; además, se

ha implementado el método ToString() por temas de pruebas y depuración.

Finalmente, la Figura 5.9 nos muestra un diagrama de esta clase en el que se puede observar los distintos constructores mencionados anteriormente y la sobreescritura del método ToString() por temas de pruebas y depuración.

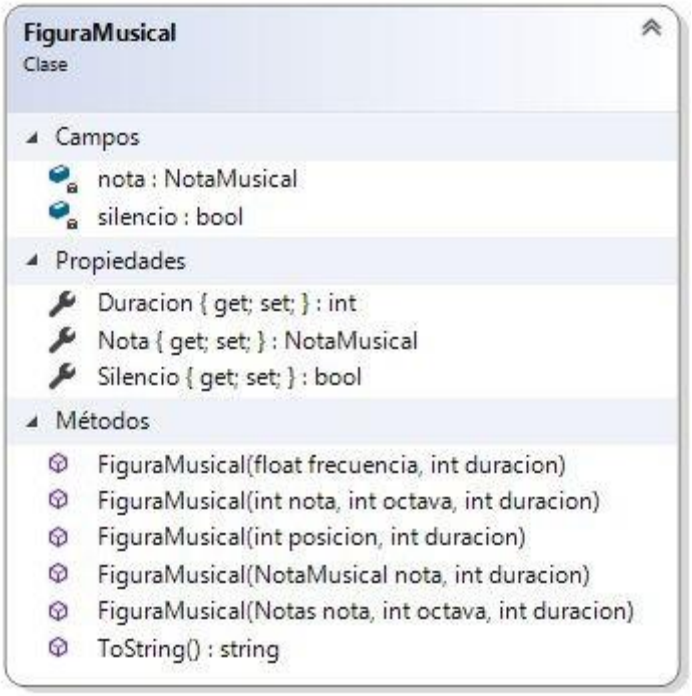

Figura 5.9: Diagrama de clase FiguraMusical.

A continuación, se va a implementar la clase Composicion, ya que Compositor hereda de esta. Esta clase dispone de varios métodos, pero la lógica principal de esta clase está formada por tres funcionalidades, añadir notas con determinada duración a una voz, cambiar el instrumento de una voz y guardar el archivo MIDI en disco.

Con esta clase ya empezamos a trabajar con la librería DryWetMidi, al probar dicha librería nos damos cuenta de que hay un problema, no se registra bien los cambios de instrumentos; notificamos el error al autor de la librería y nos comunica que lo arreglarán en la próxima versión, además nos proporciona un script para poder solucionar el problema, lo que corresponde al método ArregloBug. Veamos entonces la implementación de los principales métodos y la del constructor, en este último es donde se inicializan las variables.

```
public Composicion(int tempo = 120, int voces = 1)
{
  if (tempo < 0) throw new Exception("Tempo no puede ser negativo");
  if (voces < 1) throw new Exception("Como mínimo tiene que haber 1 voz");
  midiFile = new MidiFile();
  this.tempo = Melanchall.DryWetMidi.Smf.Interaction.Tempo.FromBeatsPerMinute(tempo);
  patterns = new PatternBuilder[voces];
  for (int i = 0; i < voces; i++)
  {
    patterns[i] = new PatternBuilder();
  }
}

public void CambiarInstrumento(GeneralMidiProgram instrumento, int voz = 1)
{
  patterns[voz - 1].SetProgram(instrumento);
}

//voz 1º, 2º, 3º.....
public void AnadirNota(NotaMmusical nota, ITimeSpan duracion = null, int voz = 1)
{
  if (duracion == null) duracion = MusicalTimeSpan.Quarter;
  if (nota != null) patterns[voz - 1].Note(ObtenerNota(nota), duracion, (SevenBitNumber)(127 - 15 *
(voz - 1)));
  else patterns[voz - 1].StepForward(duracion);
}

private Note ObtenerNota(NotaMusical nota)
{
  int octava = Mathf.Clamp(nota.Octava, Octave.MinOctaveNumber, Octave.MaxOctaveNumber - 1);
  return Note.Get((NoteName)nota.Nota, octava);
}

public void Guardar()
{
  var tempoMap = TempoMap.Create(tempo);
  foreach (var pattern in patterns)
  {
    var trackChunk = pattern.Build().ToTrackChunk(tempoMap);
    ArregloBug(trackChunk);
    midiFile.Chunks.Add(trackChunk);
  }
  midiFile.ReplaceTempoMap(tempoMap);
}

public void Guardar(string ruta, bool sobreescribir = true)
{
  Guardar();
  midiFile.Write(ruta, sobreescribir);
}

private void ArregloBug(TrackChunk trackChunk)
{
  using (var timedEventsManager = trackChunk.ManageTimedEvents())
  {
    var eventsCopy = timedEventsManager.Events.ToList();
    eventsCopy.Sort((e1, e2) =>
```

```
    {
        var timesDelta = Math.Sign(e1.Time - e2.Time);
        if (timesDelta != 0)
            return timesDelta;

        if (e1.Event is NoteEvent && !(e2.Event is NoteEvent))
            return 1;

        if (!(e1.Event is NoteEvent) && e2.Event is NoteEvent)
            return -1;

        return 0;
    });

    timedEventsManager.Events.Clear();
    timedEventsManager.Events.Add(eventsCopy);
  }
}
```

La finalidad del método ObtenerNota es limitar la octava de la nota, ya que la librería solamente admite un rango determinado de octavas. Por último, podemos ver en la Figura 5.10 el diagrama de esta clase.

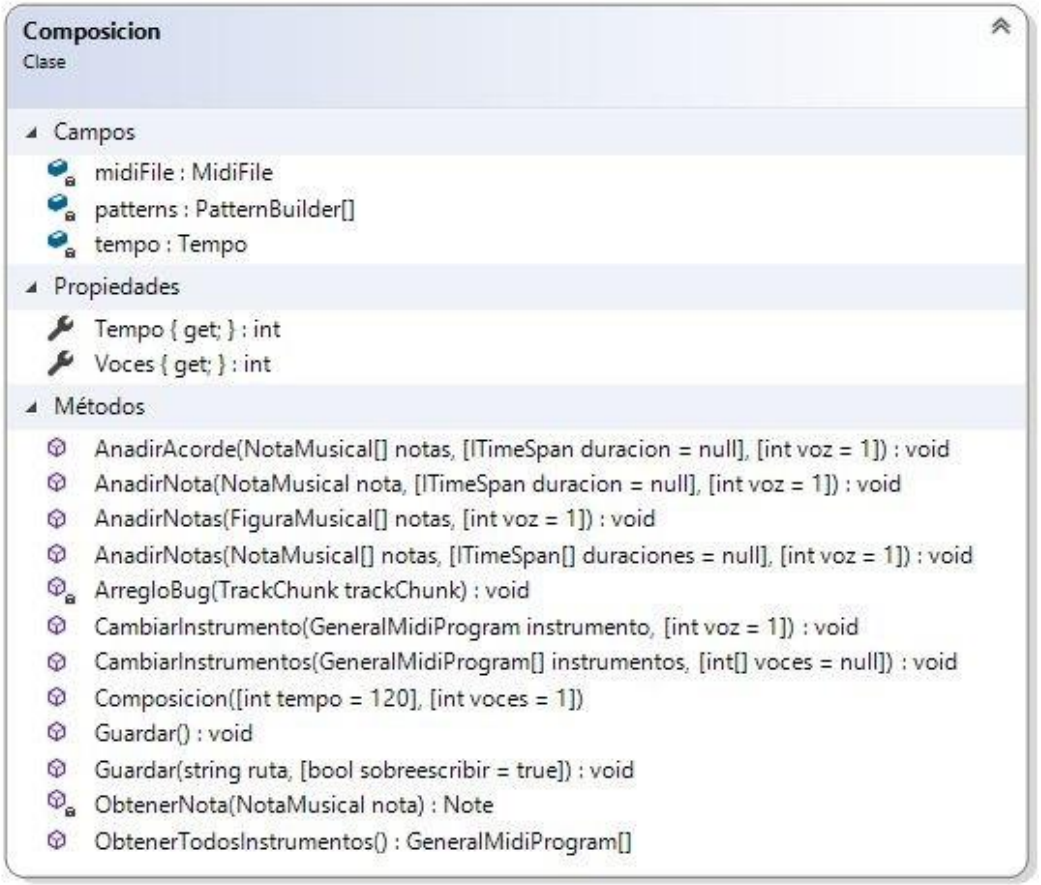

Figura 5.10: Diagrama de clase Composicion.

Para acabar, vamos a ver la clase Compositor.

Compositor hereda de Composición, por lo que comparten cierta lógica; como ya se ha explicado antes, Compositor es la encargada de crear composiciones a partir de un conjunto de muestras, por lo que contiene diversos métodos para lograr dicho objetivo.

Para facilitar el manejo de las muestras Compositor utiliza un struct llamado Frame el cual almacena un conjunto de muestras y la frecuencia de muestreo, la Figura 5.11 muestra el diagrama de este struct.

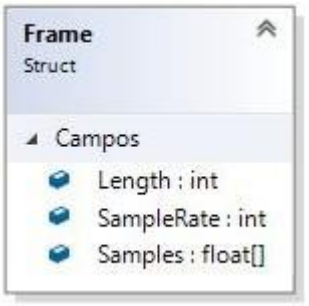

Figura 5.11: Diagrama de struct Frame.

La clase Compositor posee cuatro variables, un punto flotante que almacena el tiempo que debe durar el audio para clasificarlo (2 segundos), el clasificador, el afinador y un diccionario que almacena las relaciones entre las distintas fuentes de audio y los instrumentos disponibles.

Respecto a los métodos, se van a explicar los más relevantes.

Esta clase dispone de un método el cual recibe un conjunto de muestras y va creando la composición según las vaya analizando y clasificando, también tiene un método para obtener las figuras musicales a partir de la información obtenida del afinador, básicamente lo que hace este método es unir las notas con frecuencias similares y los silencios, de este modo en lugar de tener dos notas iguales, ambas con duración de un segundo, tendríamos una nota, con la misma frecuencia a las dos anteriores, que duraría dos segundos. A continuación, se muestra el código correspondiente a lo explicado.

```
private float tiempoClasificacion; //Tiempo de la duración del audio que se clasifica para obtener la clase a la que pertenece
private Clasificador clasificador;
```

```
private Afinador afinador;
private Dictionary<int, GeneralMidiProgram> clasesInstrumentos;

public Compositor(float tiempoClasificacion, int tempo = 120, int voces = 1) : base(tempo, voces)
{
  this.tiempoClasificacion = tiempoClasificacion;
  clasificador = new Clasificador("clasificador");
  clasificador.Cargar("Clasificador");
  afinador = new Afinador();
  clasesInstrumentos = Datos.ClasesInstrumentos;
}

public void TransformarAudio(float[] samples, int sampleRate, ref bool cancelar,
  int voces = 0, List<string> errores = null, System.Action<float> callback = null, float maxProgreso = 1)
{
  if (voces == 0) voces = Voces;
  float progreso = 0;

  try
  {
    var frames = ObtenerFrames(samples, (int)(Datos.Instance.TiempoClasificacion * sampleRate),
false);

    for (int i = 0; i < frames.Length && !cancelar; i++)
    {
      var frame = frames[i];
      TransformarFrame(frame, sampleRate);
      progreso += (maxProgreso * 0.9f) / frames.Length;
      callback?.Invoke(progreso);
    }

    if (!cancelar)
    {
      progreso = maxProgreso;
      callback?.Invoke(progreso);
    }
  }
  catch (Exception e)
  {
    errores?.Add(e.Message);
    Debug.LogError($"{e.Message} {e.StackTrace}");
  }
}

public Frame[] ObtenerFrames(float[] samples, int frameSize, bool ignorarSobrante = true)
{
  int numFrames = ignorarSobrante ? samples.Length / frameSize : Mathf.CeilToInt(samples.Length /
(float)frameSize);
  var frames = new List<Frame>();

  for (int i = 0; i < numFrames; i++)
  {
    var inicio = i * frameSize;
    var length = frameSize <= samples.Length - inicio ? frameSize : samples.Length - inicio;

    var frame = new Frame();
    frame.Samples = new float[length];
```

```
      Array.Copy(samples, inicio, frame.Samples, 0, length);
      frame.Length = frame.Samples.Length;
      frames.Add(frame);
    }

    return frames.ToArray();
}

public void TransformarFrame(Frame frame, int sampleRate)
{
    var instrumentos = ObtenerInstrumentos(frame.Samples, sampleRate);
    CambiarInstrumentos(instrumentos);
    var notas = ObtenerNotas(frame.Samples, sampleRate);
    for(int i = 0; i < notas.Length; i++) AnadirNotas(notas[i].ToArray(), i+1);
}
public List<FiguraMusical>[] ObtenerNotas(float[] samples, int sampleRate)
{
    var tempo = 60f / Tempo;
    var frameSizeIdoneo = 4096;//Es el que usa la librería Accord.net para su micrófono
    var frameSize = (int)Math.Pow(2, (int)Math.Log(frameSizeIdoneo, 2)); //FFT usa arrays con longitud
potencia de 2
    var samplesPorNota = Mathf.RoundToInt(tempo * sampleRate);
    var numeroNotas = samples.Length / samplesPorNota;
    var milisegundoPorNota = (int)(samples.Length / sampleRate * 1000f / numeroNotas);
    var figurasVoces = ObtenerNotasFFT(samples, sampleRate, frameSize, samplesPorNota, Voces);

    foreach (var figurasVoz in figurasVoces)
    {
        foreach(var figuraVoz in figurasVoz)
        {
            figuraVoz.Duracion *= milisegundoPorNota;
        }
    }

    return figurasVoces;
}


private List<FiguraMusical>[] ObtenerNotasFFT(float[] samples, int sampleRate, int frameSize, int
samplesPorNota, int numVoces = 1)
{
    var frames = ObtenerFrames(samples, frameSize);
    var voces = new List<FiguraMusical>[numVoces];
    for (int i = 0; i < voces.Length; i++)
        voces[i] = new List<FiguraMusical>();

    var notasVoces = new List<NotaMusical>[numVoces];
    for (int i = 0; i < notasVoces.Length; i++)
        notasVoces[i] = new List<NotaMusical>();

    int contSamples = 0;

    for (int i = 0; i < frames.Length; i++)
    {
        var frame = frames[i];
        var notas = afinador.ObtenerNotasFFT(frame.Samples, numVoces, sampleRate);
```

```
    for (int j = 0; j < notas.Length; j++)
    {
      notasVoces[j].Add(notas[j]);
    }

    contSamples += frame.Length;

    if (contSamples >= samplesPorNota)
    {
      for (int j = 0; j < notas.Length; j++)
      {
        var notasVoz = notasVoces[j];
        var numSilencios = notasVoz.Count(n => n == null);
        NotaMusical nuevaNota = null;
        if(numSilencios < (notasVoz.Count() - numSilencios))
        {
          nuevaNota = new NotaMusical(notasVoz.Where(n => n!= null).Select(n =>
n.Posicion).ToArray().Mode());
        }
        notasVoz.Clear();
        var voz = voces[j];
        if (voz.Count > 0)
        {
          var ultimaNota = voz.Last();
          if (ultimaNota.Nota == nuevaNota)
          {
            ultimaNota.Duracion++;
            continue;
          }
        }
        voz.Add(new FiguraMusical(nuevaNota, 1));
      }

      contSamples -= samplesPorNota;
    }
  }
  return voces;
}
```

Por último, podemos observar en la Figura 5.12 que esta clase tiene dos métodos para obtener las figuras musicales, uno con la transformada de Fourier y otro con el algoritmo de Goertzel. Esto es debido a que en esta parte se probó la de Goertzel para ver su eficacia con las muestras de un fichero, ya que un problema de la transformada de Fourier es que el tamaño del conjunto de muestras tiene que se potencia de dos (lo que impide procesar ciertos subconjuntos residuales), pero sigue careciendo de precisión a la hora de buscar múltiples frecuencias, por lo que finalmente se decide utilizar solamente el método que depende de la transformada de Fourier.

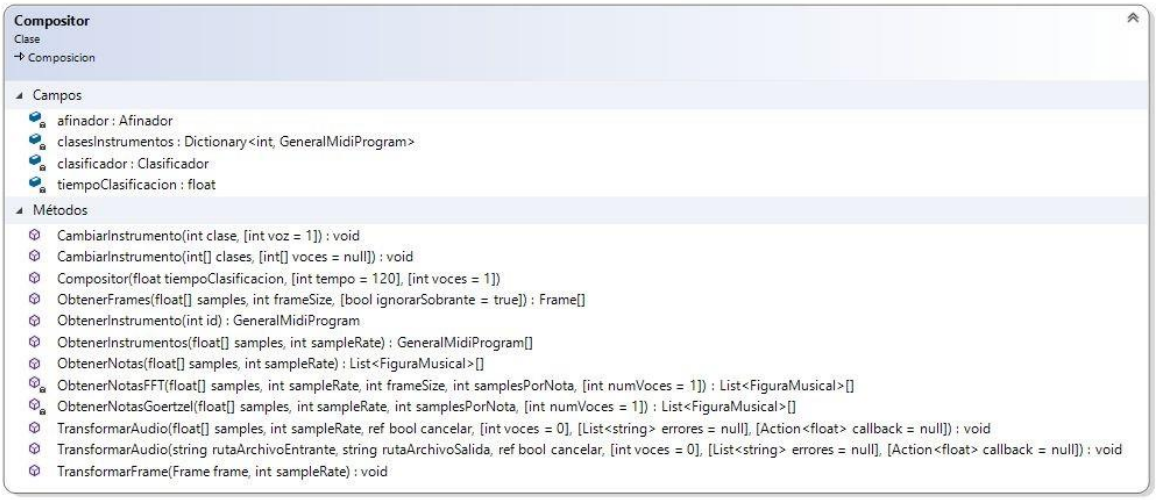

Figura 5.12: Diagrama de clase Compositor

5.4.4 Pruebas

La funcionalidad implementada en esta iteración funciona como se esperaba, para la prueba hemos seleccionado varias canciones y se la hemos pasado a nuestro sistema, la respuesta de este ha sido diversos archivos MIDI (uno por cada canción) los cuales copiaban en mayor parte los altibajos de sus respectivas canciones originales, por lo que podemos decir que ha pasado las pruebas satisfactoriamente.

5.4.5 Retrospectiva

El mayor problema (y por lo tanto retraso de tiempo) ha sido el error de la librería DryWetMidi, ya que entre intentar solucionar el error y esperar a la contestación de los autores ha habido un bloqueo del progreso del trabajo; sin embargo, al no haber más dificultades, esta iteración se ha podido completar dentro del plazo previsto. Por otra parte, aunque el sistema funcione correctamente en Linux, las distribuciones de este kernel no ofrecen nativamente un reproductor MIDI, al contrario que Windows, por lo que para poder escuchar los archivos MIDIs creados en dichas distribuciones, vamos a implementar un reproductor MIDI en la siguiente iteración.

5.5 Aplicación de los resultados

El objetivo principal de esta fase es lograr que el sistema pueda reproducir las notas obtenidas, con su respectivo timbre y duración, del procesamiento de las muestras procedentes del micrófono, de forma que ha medida que se vaya grabando el sonido, este se vaya analizando clasificando y reproduciendo concurrentemente. Además, en esta fase se pretende finalizar el desarrollo del sistema.

5.5.1 Inicialización de la iteración

En esta iteración vamos a implementar el requisito funcional RF.7 El sistema debe poder reproducir eventos MIDI. También se terminará con la implementación del requisito funcional RF.6 El sistema debe crear un archivo MIDI a partir de los datos recabados en RF.3 y RF.2, ya que, como se ha explicado antes, es necesario un reproductor MIDI para los usuarios de Linux; además, ya que la aplicación también trabaja con archivos WAVs, dicho reproductor será compatible con dichos archivos para mejorar la experiencia de usuario y no dependa de otro software de terceros. Adicionalmente, se implementarán las interfaces de usuario del sistema.

5.5.2 Diseño

Esta iteración está dividida en dos partes, por una parte, en la implementación de los requisitos funcionales RF.6 y RF.7, y por la otra, el desarrollo de las interfaces de usuario.

Para la parte de los requisitos se creará una clase base llamada Reproductor la cual contiene las operaciones básicas del reproductor del RF.6, de esta clase heredarán otras dos, AudioFilePlayer y ReproductorFicheros.

AudioFilePlayer se encargará de la reproducción de los audios WAV con la ayuda de la librería NAudioUnity, por otra parte, ReproductorFicheros es la clase encargada de manejar los dos reproductores de ficheros que hay (AudioFilePlayer y MidiFilePlayer); MidiFilePlayer la clase encargada de la reproducción de audios MIDI, esta clase ya viene con la librería MidiPlayerToolKitUnity.

Para el requisito RF.7 se utilizará la clase MidiStreamPlayer, que también la incluye dicha librería, esta clase te permite reproducir eventos MIDI, sin embargo; vamos a crear una clase llamada ReproductorStreamMidi que será la encargada, de generar los eventos MIDI a partir de la representación de notas e instrumentos de nuestro sistema y reproducirlos.

Finalmente, para la parte de las interfaces de usuario se ha decidido un diseño simple y minimalista que permita y facilite la multitarea, además de que sea susceptible a cambios. La figura 5.13 muestra un boceto de cómo será dicho diseño.

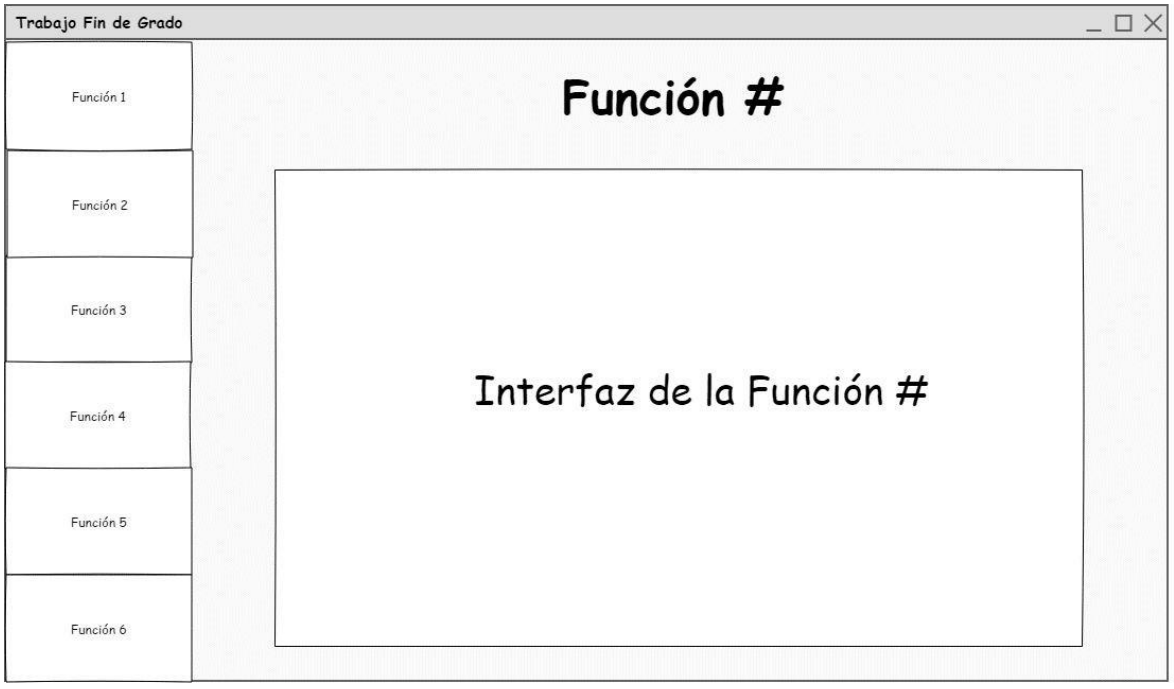

Figura 5.13: Boceto diseño interfaz de usuario.

La interfaz estaría formada por dos partes, la primera es un gran panel en el centro de la aplicación, en este panel se podrá visualizar la interfaz de la función actual, por ejemplo, el reproductor de archivos MIDI y WAV, la segunda parte es una columna de botones a la izquierda, cada botón estará vinculado a una funcionalidad del sistema, de tal forma que si el botón es pulsado se mostraría en el panel la interfaz de la funcionalidad asociada a dicho botón.
Para lograr esa interfaz de usuario se va a crear una clase base llamada

PanelFunción, de esta clase heredarán todas las demás clases que representen a una funcionalidad, además esta clase contiene una implementación base para el almacenamiento de los datos en JSON; nuestra idea es que cada panel funcional administre sus datos y que una clase llamada ManagerInterfaz sea la encargada de pedir y dar los datos a cada panel, para posteriormente guardarlos en disco y cargarlos.

Luego se creará una clase por cada funcionalidad, actualmente hay tres funcionalidades implementadas, lo que corresponde a tres paneles: PanelReproductor, PanelTransformarAudio y PanelTransformarAudioStream.
PanelReproductor es el controlador de la interfaz para el reproductor de archivos MIDI y WAV, contará por lo tanto con un controlador de volumen, barra de tiempo, botón de pausa/reproducir, botón de reiniciar y un campo de entrada de texto para indicar la ruta del audio a reproducir.
PanelTransformarAudio es el controlador de la interfaz para la transformación de archivos WAV a archivos MIDI, contará por lo tanto con campos de entrada de texto para indicar las rutas del archivo WAV entrante, el archivo MIDI saliente, el nombre del archivo creado, un botón para iniciar/detener la transformación, una barra de progreso y opciones para cambiar el valor de las variables voces y tempo empleadas en el proceso de transformación.
PanelTransformarAudioStream es el controlador de la interfaz para la transformación de audio en directo, contará por lo tanto con un botón para iniciar/detener la transformación, control de volumen, un cronógrafo y opciones para cambiar el valor de las variables voces y tempo empleadas en el proceso de transformación.

La interfaz de usuario se ha creado de forma gráfica mediante el editor de Unity.

5.5.3 Implementación

Comencemos explicando la implementación de los requisitos funcionales RF.6 y RF.7.
Tal y como se explica en la parte de diseño, creamos una clase abstracta llamada Reproductor, la Figura 5.14 muestra el diagrama de esta clase.

Figura 5.14: Diagrama de clase Reproductor.

Como podemos ver la clase implementa los métodos básicos y propiedades básicas de un reproductor, pausar, reproducir, reiniciar, detener, volumen, progreso (porcentaje del tiempo de la canción reproducido, se obtiene a partir de TiempoActual y TiempoTotal, tiempo reproducido y duración del audio respectivamente) y una variable booleana para saber si está pausado o no el reproductor.

De esta clase abstracta heredan las clases ReproductorFicheros y AudioFilePlayer, ambas, además de compartir las características heredadas de su clase padre, contienen operaciones y variables específicas para lograr su función, por ejemplo, AudioFilePlayer se ayuda de un struct para manejar las muestras de los audios.

También existe otro reproductor, MidiExternalFilePlayer, este hereda de MidiFilePlayer (esta clase pertenece a la librería MidiPlayerToolKitUnity); la clase MidiExternalFilePlayer únicamente contiene una variable y un método, ya que esta clase tiene como función aprovechar la lógica de la clase padre para poder reproducir archivos MIDI que se encuentren fuera del proyecto, ya que el reproductor que viene con la versión gratuita de la librería carece de dicha funcionalidad. Finalmente recalcar que la clase ReproductorFicheros es la clase

responsable de manejar la reproducción de un audio según el formato de este, por lo que contiene dos referencias, una a la clase AudioFilePlayer y otra a MidiExternalFilePlayer. Las Figuras 5.15, 5.16 y 5.17 muestran los diagramas de las clases ReproductorFicheros, AudioFilePlayer y MidiExternalFilePlayer respectivamente.

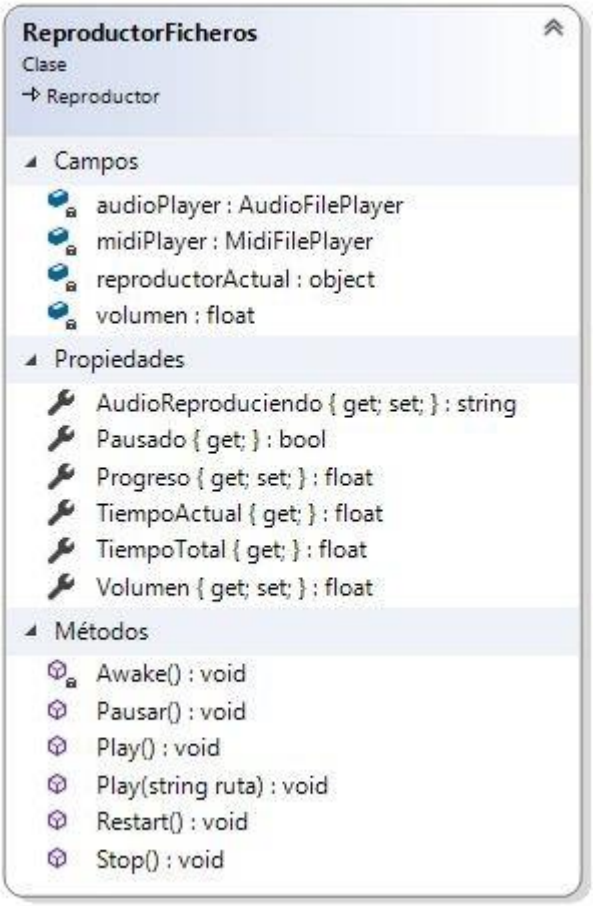

Figura 5.15: Diagrama de clase ReproductorFicheros.

Figura 5.16: Diagrama de clase AudioFilePlayer.

Figura 5.17: Diagrama de clase MidiExternalFilePlayer.

Para la reproducción de eventos MIDI necesaria para el requisito funcional RF.7 necesitamos una clase que se llamará ReproductorStreamMidi, será la encargada de interpretar nuestros datos y transformarlos a eventos MIDI con el fin de reproducirlos. Esta clase tendrá una corrutina para manejar los eventos de las distintas voces mediante la duración de cada uno de estos, dichos eventos serán interpretados por la clase MidiStreamPlayer, la cual forma parte de la librería utilizada. La Figura 5.18 muestra el diagrama de la clase anterior descrita, posteriormente se expone la implementación de dicha corrutina.

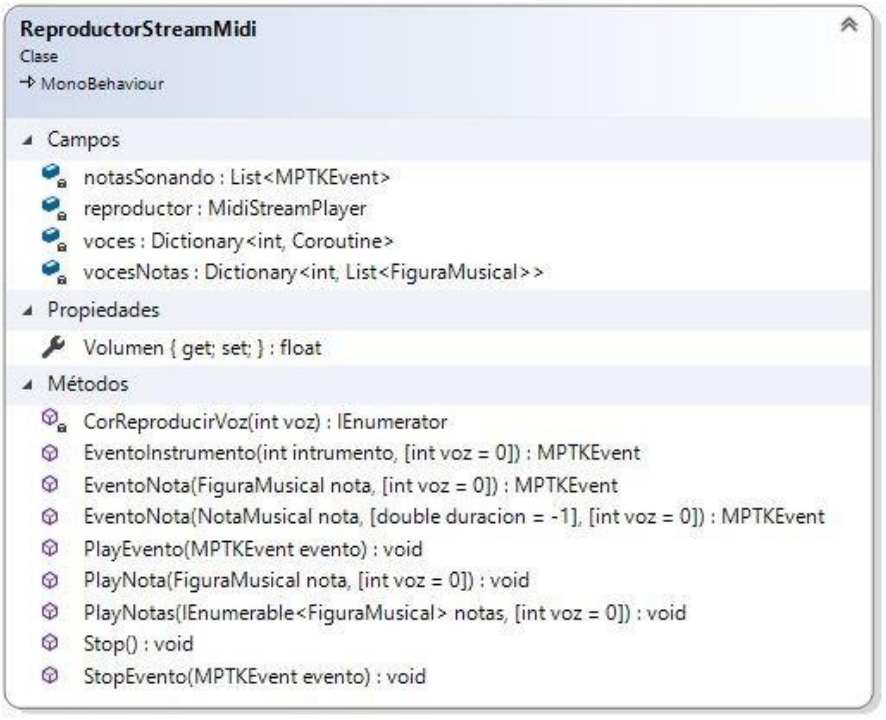

Figura 5.18: Diagrama de clase ReproductorStreamMidi.

```
IEnumerator CorReproducirVoz(int voz)
{
  var notas = vocesNotas[voz];

  var tiempo = 0f;
  MPTKEvent notaEvento = null;
  while (notas.Count > 0)
  {
    if (notaEvento != null)
    {
      notasSonando.Remove(notaEvento);
    }

    var figura = notas[0];
    var nota = figura.Nota;

    notaEvento = EventoNota(figura, voz);
    PlayEvento(notaEvento);
    notasSonando.Add(notaEvento);
    tiempo += figura.Duracion < 0 ? float.MaxValue : figura.Duracion / 1000f;
    yield return new WaitUntil(() => (tiempo -= Time.deltaTime) <= 0);

    notas.RemoveAt(0);
  }

  vocesNotas.Remove(voz);
  voces.Remove(voz);
}
```

Pasemos ahora al desarrollo de la interfaz gráfica de usuario.

La primera clase a crear es la clase abstracta PanelFuncion, la cual es la clase base de todos los paneles de funciones. Esta clase provee de métodos para la serialización y deserialización de los datos almacenados en cada panel, dichos datos, por ejemplo, el volumen o el número de voces, se guardarán en formato JSON gracias a la librería JSON.NET. La Figura 5.19 muestra el diagrama de esta clase.

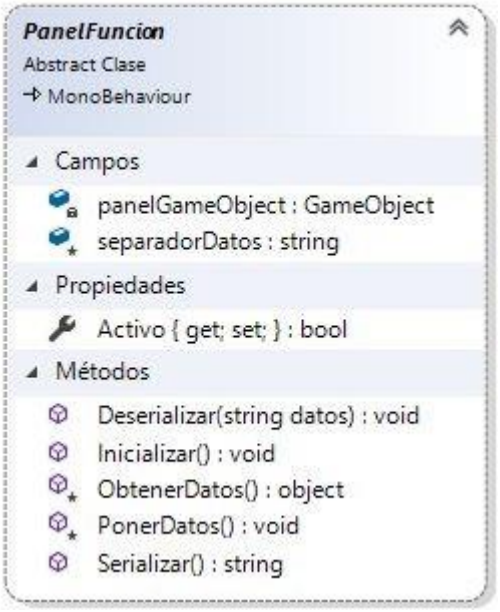

Figura 5.19: Diagrama de clase PanelFuncion.

Todas las instancias de tipo PanelFuncion serán administradas por la clase ManagerInterfaz. Esta clase, que hereda de PanelFunción, es la encargada de manejar el funcionamiento de los distintos paneles para cada funcionalidad presente en el sistema. Para llevar a cabo dicha administración, la clase cuenta con un struct en el cual se asocia cada botón (representado con la clase BotonFuncion) con su panel correspondiente. Por lo tanto, esta clase recibirá el evento del click de cada botón y según el botón pulsado mostrará el panel correspondiente, además esta clase administradora cuenta con métodos para guardar en disco y cargar del mismo los datos de todas las instancias de tipo PanelFuncion, para lograr dicha función se ayuda de una clase estática llamada SistemaGuardado, la cual cuenta con métodos para la lectura y escritura de ficheros. Las Figuras 5.20, 5.21 y 5.22 muestran los diagramas de las clases ManagerInterfaz, BotonFuncion y SistemaGuardado respectivamente.

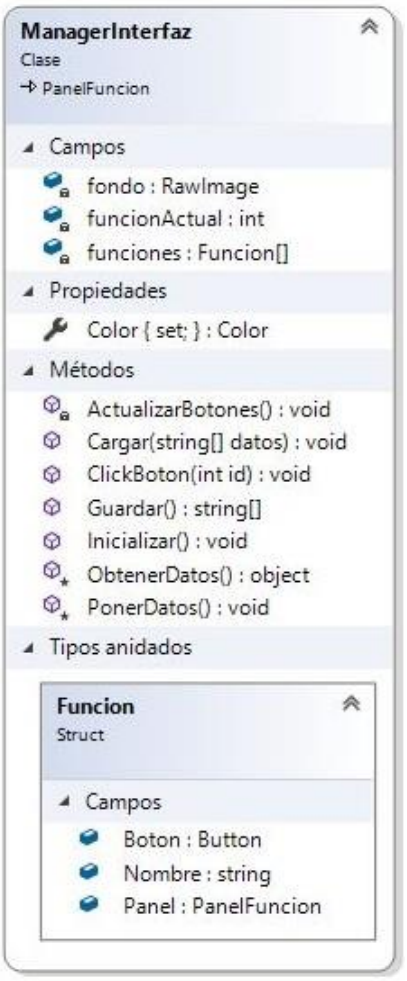

Figura 5.20: Diagrama de clase ManagerInterfaz.

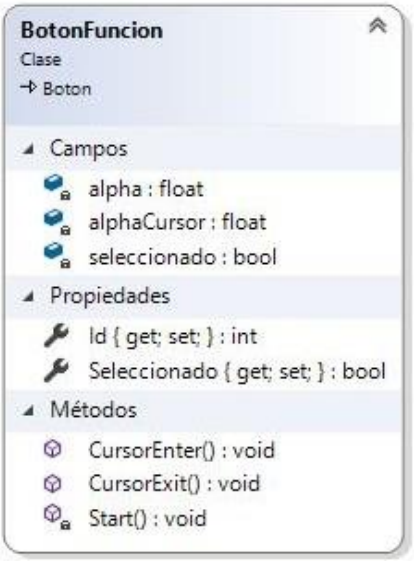

Figura 5.21: Diagrama de clase BotonFuncion.

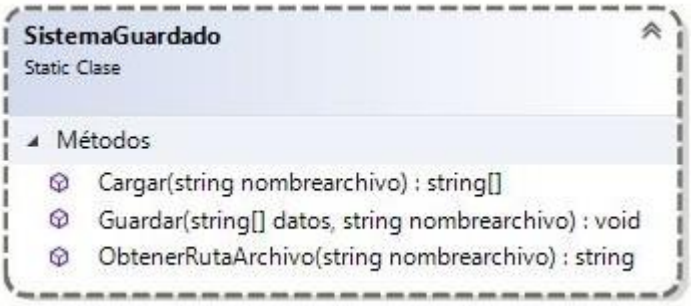

Figura 5.22: Diagrama de clase SistemaGuardado.

Ya tenemos implementada la estructura principal de la interfaz de usuario, ahora nos queda crear los distintos paneles para cada funcionalidad. Aunque en la parte de diseño se mencionó la existencia de tres paneles, finalmente habrá cuatro. El primer panel es para el reproductor de archivos de audio, el segundo reproductor corresponde al transformador de audios registrado, el tercer panel está asignado al transformador de audio en directo, y finalmente el cuarto panel se destinará a la configuración global del sistema. Debido a que el primer y el tercer panel contarán con control de volumen, vamos a crear una clase abstract llamada PanelFuncionMuteable, la cual es hija de PanelFuncion y tendrá las operaciones y variables necesarias para la implementación del control de volumen, la Figura 5.23 muestra el diagrama de dicha clase.

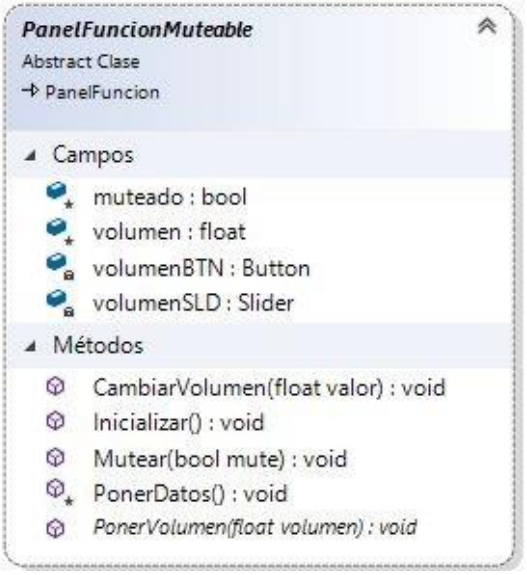

Figura 5.23: Diagrama de clase PanelFuncionMuteable.

A continuación, veamos la implementación de cada panel.

Empezamos con el primer panel, el cual pertenece al reproductor de audios. La clase correspondiente a manejar este panel se llama PanelReproductor, esta clase cuenta con una referencia a una instancia de ReproductorFicheros para poder reproducir los audios, además cuenta con la lógica de control del volumen y la barra de tiempo; además, contará con un texto que indicará al usuario el estado del reproductor y dará la opción de mostrar al usuario un explorador de archivos para poner la ruta del audio más fácilmente, dicho explorador de archivo es proporcionado por la librería UnitySimpleFileBrowser. Las Figuras 5.24 y 5.25 muestran el diagrama de PanelReproductor y la vista de dicho panel.

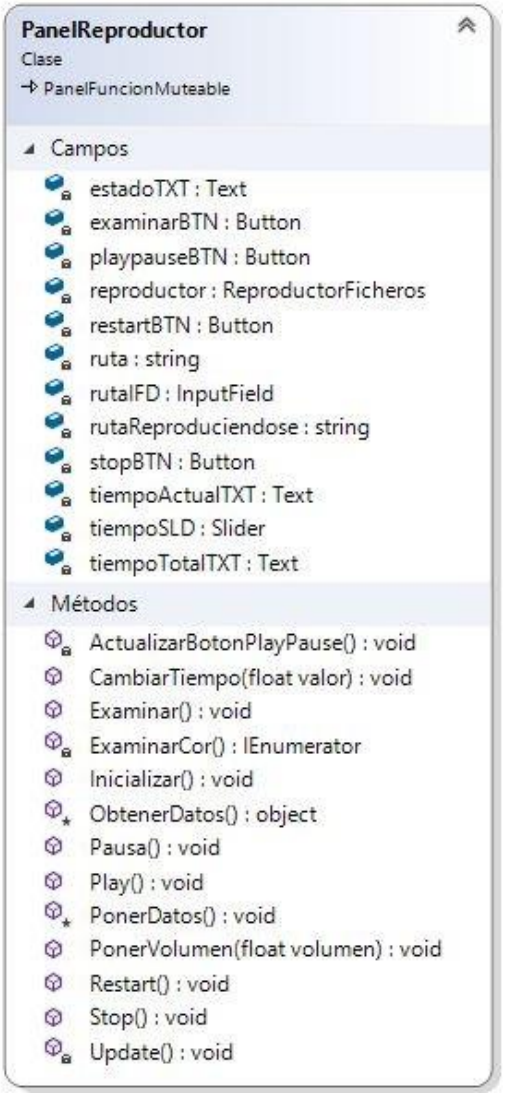

Figura 5.24: Diagrama de clase PanelReproductor.

Figura 5.25: Vista de PanelReproductor.

Continuamos con el segundo panel, este panel está asignado al transformador de audios registrados, el cual es administrado por la clase PanelTransformarAudio. Esta clase cuenta con una referencia a una instancia de Compositor para poder transformar los archivos de audio con formato WAV, además de administrar los campos de entrada de usuario y la barra de proceso. La transformación se realiza en una hebra con el fin de no bloquear el ciclo de vida de Unity, además se utiliza una llamada de retorno (callback) para actualizar la barra de proceso. A continuación, se muestra la implementación de la corrutina encargada de la transformación y seguidamente las Figura 5.26 y 5.27, las cuales muestran el diagrama de esta clase y la vista de este panel respectivamente.

```
IEnumerator CorTransformar()
{
  var entrada = rutaArchivoEntradaIFD.text;
  var salida = System.IO.Path.Combine(rutaArchivoSalidaIFD.text);
  float progreso = 0;
```

```
  var compositor = new Compositor(Datos.Instance.TiempoClasificacion, tempo, voces);
  var errores = new List<string>();
  var hebra = new Thread(() => compositor.TransformarAudio(entrada, salida, ref cancelar, voces,
errores, (x) => progreso = x));
  hebra.Priority = System.Threading.ThreadPriority.Highest;
  hebra.Start();
  while (hebra.IsAlive)
  {
    PonerProgreso(progreso);
    yield return new WaitForFixedUpdate();
  }

  if (errores.Count > 0)
  {
    var error = $"Error al transformar el audio {entrada}: {string.Join("\n", errores)}";
    ErrorManager.Instance.CrearError(error);
    Debug.Log(error);
  }

  Utilidades.PonerPortaPapeles(salida);
  PonerProgreso(progreso);
  PonerPlayPause(true);

  yield return null;
  hebra.Join();
}
```

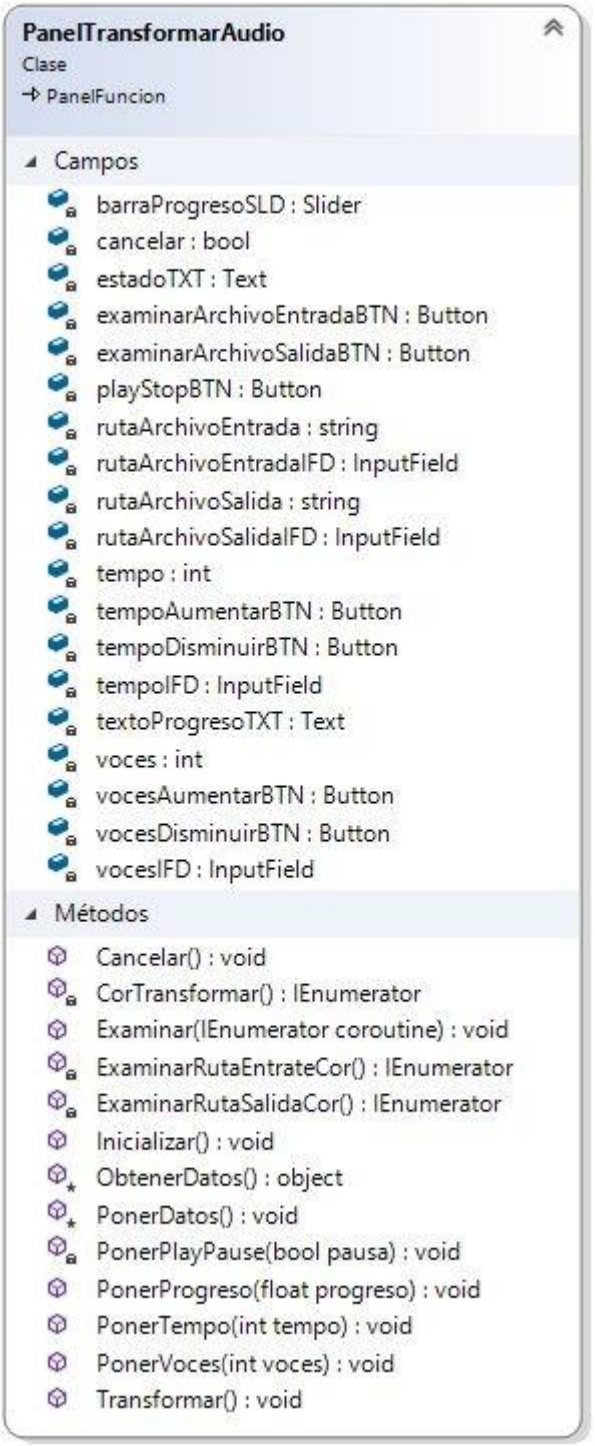

Figura 5.26: Diagrama de clase PanelTransformarAudio.

Figura 5.27: Vista de PanelTransformarAudio.

Seguidamente, implementamos la clase PanelTransformarAudioStream, esta clase se encarga de la lógica del tercer panel. Este panel se dedica a transformar el audio captado por el micrófono y posteriormente, reproducir el resultado de dicha transformación. La clase PanelTransformarAudioStream, al igual que PanelReproductor, hereda de PanelFuncionMuteable, ya que cuenta con un control de volumen. La lógica principal de la clase PanelTransformarAudioStream está compuesta por tres corrutinas, una para crear el Compositor, otra para activar el micrófono y la última para reproducir el resultado a través de la clase ReproductorStreamMidi. A continuación, se muestra la implementación de estas tres corrutinas y las Figuras 5.28 y 5.29, las cuales corresponden al diagrama de la clase PanelTransformarAudioStream y la vista del panel respectivamente.

```
IEnumerator CorTransformarStream()
{
  if (Microphone.devices.Length == 0) //Si no detecta ningún micrófono
  {
    ErrorManager.Instance.CrearError("No hay micrófono conectado");
```

```
      yield break; //Terminamos la coroutina
    }
    var sampleRate = 4096 * 10;
    var lengthSec = Mathf.RoundToInt(Datos.Instance.TiempoClasificacion);

    compositor = new Compositor(Datos.Instance.TiempoClasificacion, tempo, voces);

    var errores = new List<string>();
    var samples = new List<float>();

    StartCoroutine(GrabarAudio(samples, lengthSec, sampleRate));
    StartCoroutine(AnalizarSamples(samples, sampleRate));
}

IEnumerator GrabarAudio(List<float> samples, int lengthSec, int frequency, bool loop = false)
{
    PonerContadorTiempo(0);
    while (true)
    {
        var audio = Microphone.Start(null, loop, lengthSec, frequency);

        while (Microphone.IsRecording(null))
        {
            PonerContadorTiempo(contadorTiempo + Time.deltaTime);
            yield return null;
        }
        var samplesAudio = new float[audio.samples];
        audio.GetData(samplesAudio, 0);
        samples.AddRange(samplesAudio);
    }
}

IEnumerator AnalizarSamples(List<float> allSamples, int sampleRate)
{

    while (true)
    {
        yield return new WaitUntil(() => allSamples.Count > 0);
        var samples = allSamples.ToArray();
        allSamples.Clear();
        var instrumentos = compositor.ObtenerInstrumentos(samples, sampleRate);
        var notasVoces = compositor.ObtenerNotas(samples, sampleRate);
        for (int i = 0; i < instrumentos.Length; i++)
reproductor.PlayEvento(reproductor.EventoInstrumento((int)instrumentos[i]));
        for (int i = 0; i < notasVoces.Length; i++) reproductor.PlayNotas(notasVoces[i]);
    }
}
```

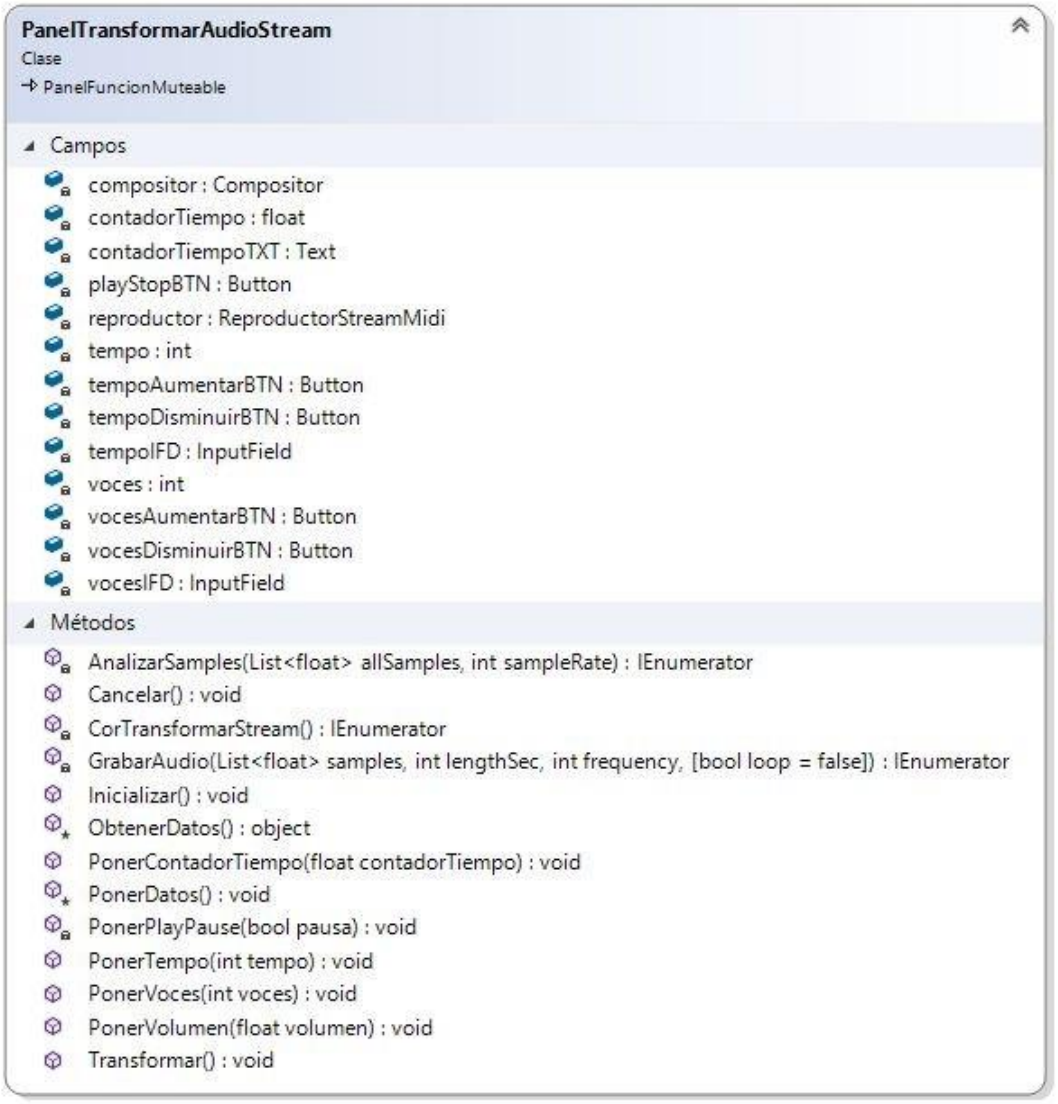

Figura 5.28: Diagrama de clase PanelTransformarAudioStream.

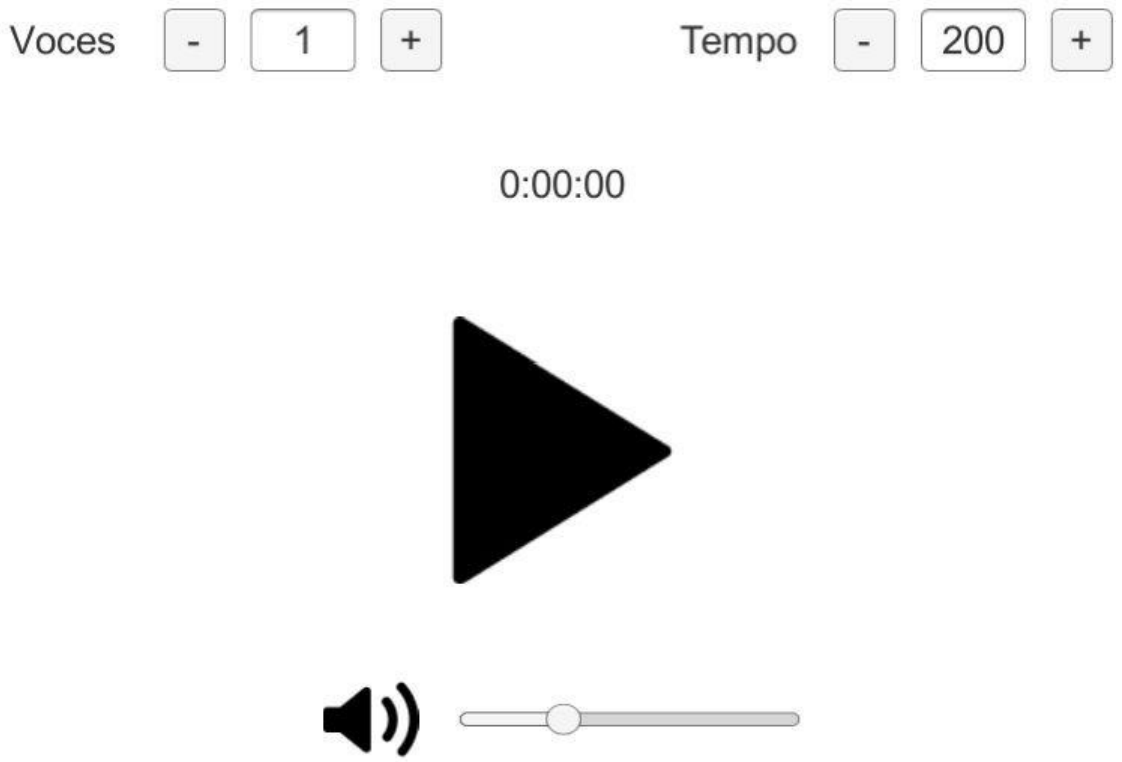

Figura 5.29: Vista de PanelTransformarAudioStream.

Por último, vamos a implementar la clase correspondiente al cuarto panel. Esta clase, llamada PanelOpciones, contiene métodos para permitir al usuario modificar las relaciones entre fuente del audio e instrumentos y el color de fondo de la aplicación. A la hora de cambiar las relaciones el usuario puede escuchar el sonido de cada instrumento disponible para elegir el de su agrado, debido a esto, la clase cuenta con un control de volumen y, por lo tanto, hereda de PanelFuncionMuteable y tiene una referencia a una instancia de ReproductorStreamMidi. Además, debido a la complejidad de la lógica de la interfaz de la relación entre fuentes e instrumentos, ha sido necesario crear una clase especial llamada BotonInstrumento para captar los diversos eventos del ratón, esta clase hereda de la clase Button, la cual es nativa de Unity. Además, cabe mencionar que en este panel se puede visualizar los créditos de la aplicación, en dichos créditos se muestran las distintas librerías que conforman el sistema y los recursos visuales utilizados, con la posibilidad de ir a las páginas oficiales de cada elemento, esta última funcionalidad está implementada en la clase Link. Para acabar, se exponen la implementación de la corrutina encargada de la prueba de los instrumentos y las Figuras 5.30, 5.31, 5.32 y 5.33,

las cuales muestran el diagrama de clase de PanelOpciones, BotonInstrumento, Link y la vista del panel respectivamente.

```
IEnumerator ProbarInstrumentoCoroutine(int instrumento)
{
  NotaMusical notaPrueba = notaPruebaBase;
  var eventoInstrumento = reproductor.EventoInstrumento(instrumento);
  reproductor.PlayEvento(eventoInstrumento);
  var eventoNotaProbando = reproductor.EventoNota(notaPrueba);
  reproductor.PlayEvento(eventoNotaProbando);

  float posicionNota = notaPrueba.Posicion;
  float escala = 0.5f;
  listaClaseInstrumentos.GetComponentInParent<ScrollRect>().enabled = false;
  listaInstrumentos.GetComponentInParent<ScrollRect>().enabled = false;
  while (probando)
  {
    var input = Input.mouseScrollDelta.y;
    posicionNota += input * escala;
    if (Math.Abs(notaPrueba.Posicion - posicionNota) >= 1)
    {
      reproductor.StopEvento(eventoNotaProbando);
      notaPrueba = new NotaMusical((int)posicionNota);
      eventoNotaProbando = reproductor.EventoNota(notaPrueba);
      reproductor.PlayEvento(eventoNotaProbando);
    }

    yield return null;
  }
  listaClaseInstrumentos.GetComponentInParent<ScrollRect>().enabled = true;
  listaInstrumentos.GetComponentInParent<ScrollRect>().enabled = true;
  reproductor.StopEvento(eventoNotaProbando);
}
```

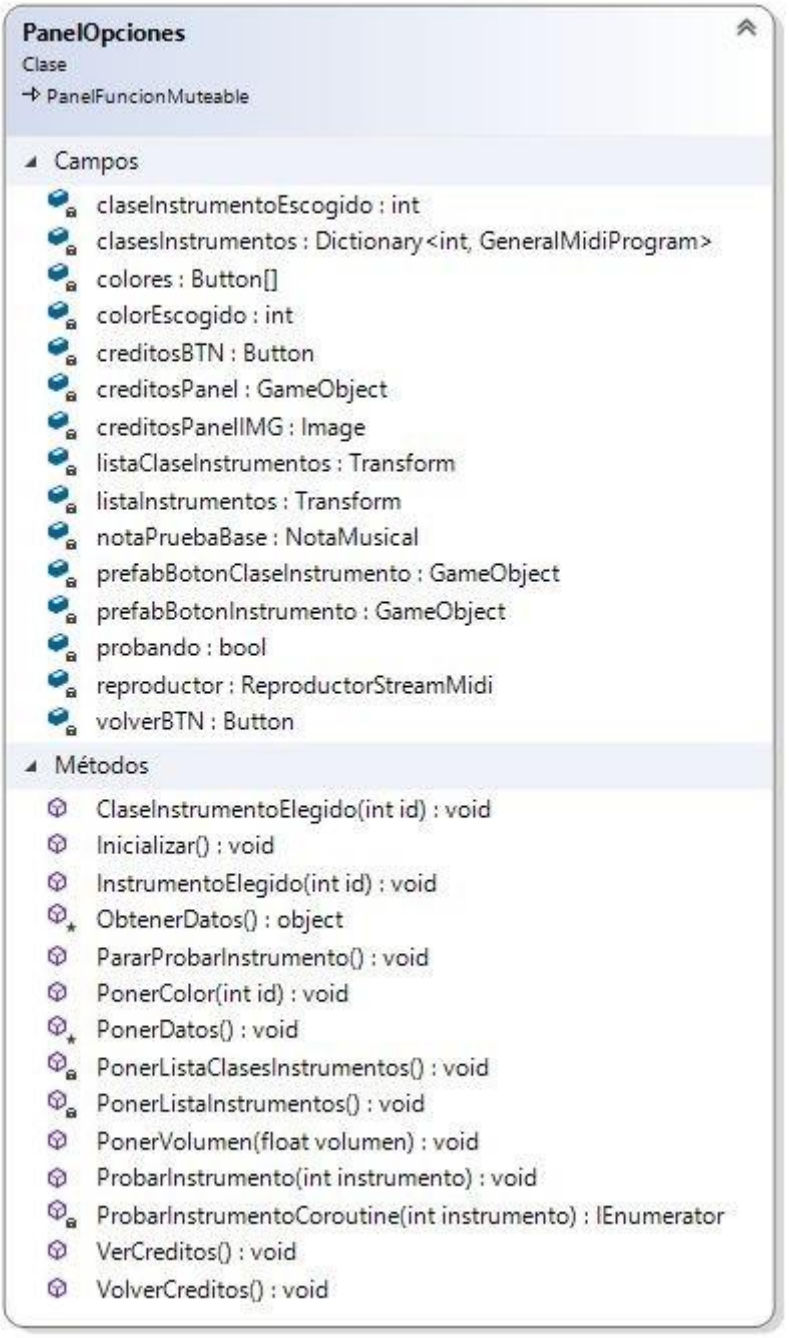

Figura 5.30: Diagrama de clase PanelOpciones.

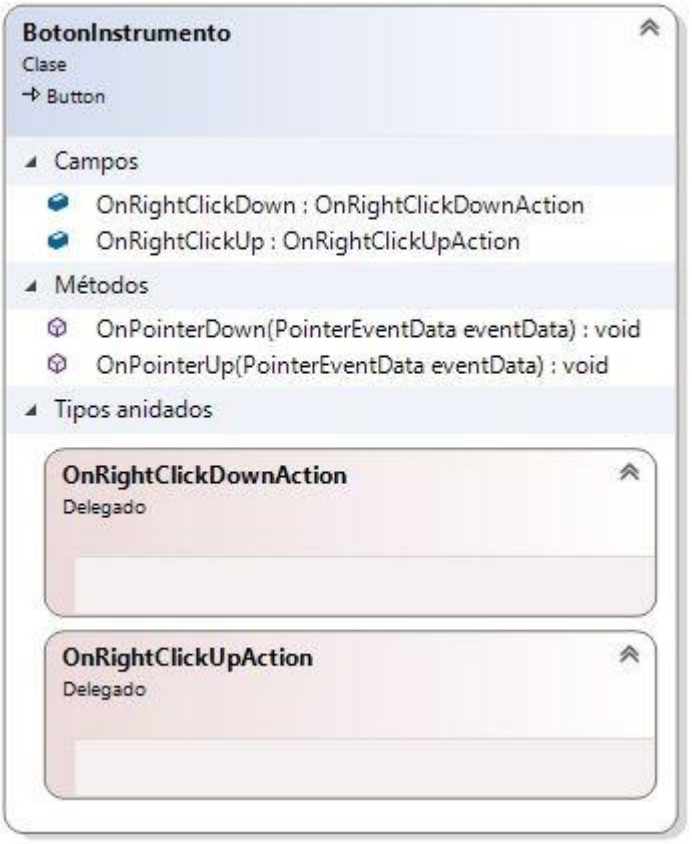

Figura 5.31: Diagrama de clase BotonInstrumento.

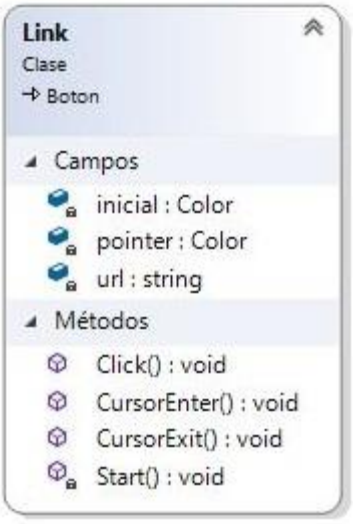

Figura 5.32: Diagrama de clase Link.

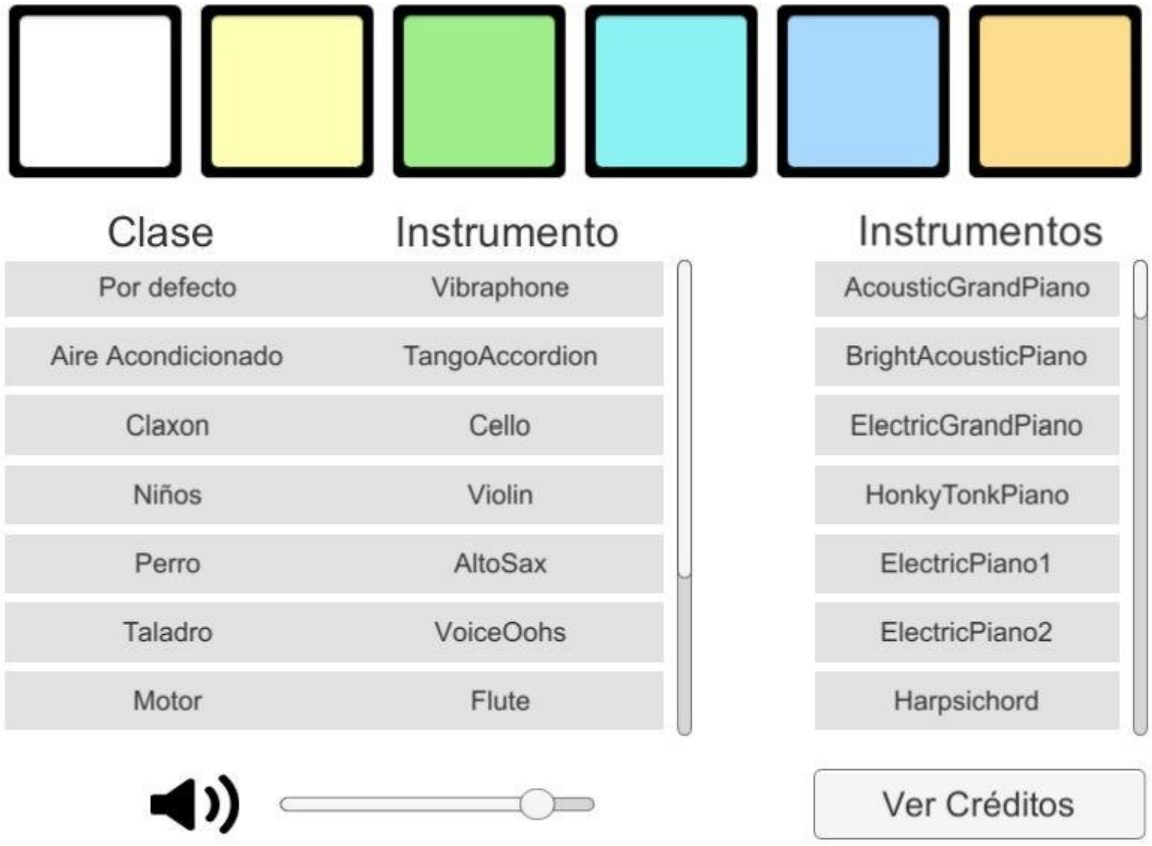

Figura 5.33: Vista de PanelOpciones.

La clase ErrorManager, que se ha podido observar en las implementaciones de algunos paneles, es la encargada de crear una ventana para mostrar los errores al usuario. La Figura 5.34 muestra el diagrama de esta clase.

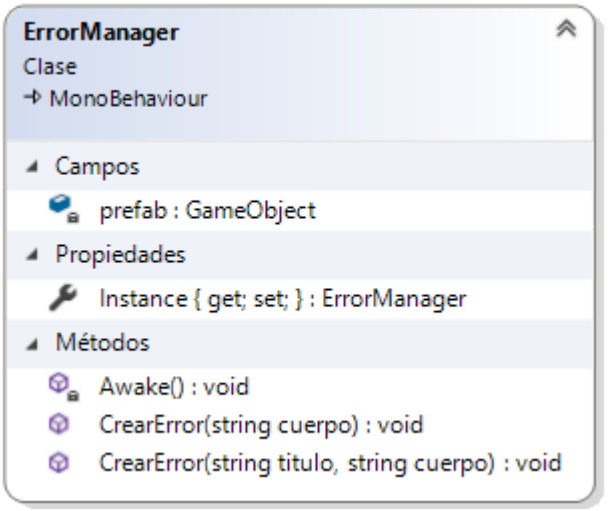

Figura 5.34: Diagrama de clase ErrorManager.

5.5.4 Pruebas

Se ha probado cada una de las distintas funcionalidades tanto en Windows como en Linux y no ha habido problemas. Estas pruebas finales se han realizado con intención de comprobar el correcto funcionamiento de todo el sistema, así como su tiempo de respuesta a la hora de emplear las diversas funcionalidades al mismo tiempo. Dichas pruebas se han realizado directamente a través de la interfaz de usuario. El resultado de las pruebas ha sido favorable, el sistema respondió a las peticiones realizadas correctamente, por lo que podemos concluir el desarrollo del mismo.

5.5.5 Retrospectiva

Esta iteración ha sido la más densa respecto a las demás iteraciones en cuanto a resultados obtenidos y código escrito, no esperábamos tal magnitud de tiempo empleado en esta iteración debido a que finalmente se ha implementado más funcionalidades de las previstas con el fin de mejorar la experiencia de usuario y cumplir con las retribuciones de los autores de los distintos recursos de terceros empleados. Por lo tanto, el tiempo dedicado a esta iteración ha excedido al planeado; sin embargo, gracias a que en algunas iteraciones anteriores se realizaron en un tiempo menor a lo planificado, el tiempo de desarrollo del proyecto en su conjunto está dentro de lo programado.

6

Conclusiones

6.1 Conclusiones

Gracias a la realización de este proyecto, hemos obtenidos bastantes conocimientos sobre ramas que, aunque algunas están vinculadas a la carrera, otras no lo están.

Para comenzar, hemos tenido que documentarnos sobre algoritmos usados en teoría de señales, especialmente la transformada de Fourier con la información obtenida de *https://es.wikipedia.org/wiki/Transformada_de_Fourier*. Además, hemos obtenido conocimientos sobre las propiedades del sonido a partir del enlace *https://es.wikipedia.org/wiki/Sonido#Propiedades*, incluyendo también conocimientos generales sobre formatos de audio obtenidos de *https://es.wikipedia.org/wiki/Audio_digital.* Haciendo mención a esto último, aparte de obtener dichos conocimientos generales, para este proyecto ha sido necesario formarse especialmente en los formatos WAV y MIDI. Además, hemos aprendido cómo se obtienen las distintas frecuencias de cada una de las notas musicales a través de la página web

Continuando, en este proyecto se ha empleado una metodología de desarrollo que no se ha empleado en la carrera, PXP; hemos tenido que informarnos sobre cómo emplear dicha metodología, conocer sus características así como sus ventajas e inconvenientes, todo esto a través de los enlaces *https://es.wikipedia.org/wiki/Programaci%C3%B3n_extrema* y *https://www .researchgate.net/publication/229046039_Personal_Extreme_Programming-An_Agile_Process_for_ Autonomous_Developers..* A parte de esto, nos hemos documentado sobre los métodos ágiles en general con la información obtenida de *https://www.heflo.com/es/definiciones/metodologia-agil.*

Con este proyecto también hemos aprendido programación básica en Python y aumentar los conocimientos que teníamos sobre Unity y C#, ya que hemos tenido que recurrir a hebras y corrutinas para impedir bloquear el ciclo de ejecución del motor, intentando optimizar al máximo el tiempo empleado en las distintas funcionalidades utilizando los recursos disponibles. También nos hemos informado cómo funciona internamente .NET y, por lo tanto, Mono, gracias a la documentación obtenida de *https://es.wikipedia.org/wiki/Microsoft_.NET.*

Respecto a la parte del entrenamiento del clasificador de audios, hemos comprendido cómo funcionan distintos tipos de algoritmos de clasificación y su rendimiento según su finalidad, por lo que esto nos ha servido para obtener experiencia con distintos algoritmos básicos empleados en la inteligencia artificial.

Por último, al guardar los datos de la aplicación en formato JSON, se ha conseguido tener un manejo amplio sobre dicho formato y cómo emplearlo en el lenguaje C#. Otro formato el cual también se ha conseguido bastante experiencia a la hora de trabajarlo con el mismo lenguaje de programación es el formato MIDI, ya que a la hora de crear los audios con dicho formato, hemos comprendido cómo funciona internamente, lo que puede ofrecer y de lo que carece, ya que, por ejemplo, al ser un formato basado en eventos, para poder reproducirlos se necesita un dispositivo que interprete y reproduzca dichos eventos, a diferencia de otros formatos como WAV.

Finalmente, mencionar que, aunque en un principio no se iba a utilizar, el uso de Unity ha facilitado la tarea de conseguir una aplicación compatible con múltiples sistemas operativos.

6.2 Trabajo futuro

Con vista al futuro, el sistema actual podría modificarse con el fin de hacerlo más sensible a la hora de captar el sonido ambiente y también para compartir la lógica con otras aplicaciones.

Para hacerlo más sensible en la captación se puede extender la actual lógica de la transformación del sonido para que pueda captar los cambios de ritmos presentes en el audio, y que el sonido resultante de dicha transformación adapte la duración de cada nota al ritmo correspondiente.

Para la parte de la distribución de la lógica, la idea sería crear un servidor bajo un servicio PaaS de terceros. La aplicación de dicho servidor tendría implementada toda la lógica de la transformación, de esta forma cualquier aplicación, sea propia o de terceros, podría transformar sus propios audios mediante peticiones REST. Por último, también estaría bien implementar la lectura de otros tipos de formatos de audio, con el fin de hacer que el sistema sea compatible con otros formatos de audio.

Bibliografía

Justin Salomon, Christopher Jacoby, Juan Pablo Bello (2014). En Proceedings of the 22nd ACM international conference on Multimedia. pp 1041- 1044. A Dataset and Taxonomy for Urban Sound Research.

Transformada de Fourier. En Wikipedia. Consultado en 2019 de *https://es.wikipedia.org/wiki/Transformada_de_Fourier.*

Propiedades del sonido. En Wikipedia. Consultado en 2019 de *https://es.wikipedia.org/wiki/Sonido#Propiedades.*

Audio Digital. En Wikipedia. Consultado en 2019 de *https://es.wikipedia.org/wiki/Audio_digital.*

Cálculo de la frecuencia de las notas musicales. En El club del autodidacta. Consultado en 2019 de *http://elclubdelautodidacta.es/wp/2012/08/calculo-de-la-frecuencia-de-nuestras-notas-musicales.*

Programación Extrema. En Wikipedia. Consultado en 2019 de *https://es.wikipedia.org/wiki/Programaci%C3%B3n_extrema.*

Programación Extrema para desarrolladores autónomos. En Researchgate. Consultado en 2019 de *https://www.researchgate.net/publication /229046039_Personal_Extreme_Programming-An_Agile_Process_for_ Autonomous_Developers.*

Metodología ágil. En Wikipedia. Consultado en 2019 de *https://www.heflo.com/es/definiciones/metodologia-agil.*

Microsoft .NET Framework. En Wikipedia. Consultado en 2019 de *https://es.wikipedia.org/wiki/Microsoft_.NET.*

Anexo A

Manual de Usuario

En este apéndice se describe un manual de usuario del sistema desarrollado. En dicho manual se muestran las distintas vistas que componen el sistema, y se explica las distintas funcionalidades con las que puede interactuar el usuario y cómo se realiza dicha interacción. Este manual tiene el objetivo de ayudar, como guía, a cualquier usuario que utilice el sistema.

A.1 Requisitos e instalación

La aplicación presenta dos ejecutables, uno para sistemas Windows y el otro para sistemas Linux; además, para poder disfrutar de todas las funcionalidades del sistema, es necesario tener conectado al equipo informático dispositivos de entrada y salida de audio. Debido a que el sistema funciona sobre un motor gráfico, es recomendable no ejecutarlo sobre una máquina virtual, ya que puede impedir la aceleración gráfica y, en ese caso, el programa podría no ejecutarse correctamente.

Al ejecutar por primera vez la aplicación nos aparece la vista representada en la Figura A.1. Como podemos ver, en el lado izquierdo hay una lista de botones y en el lado derecho el panel de la función actualmente activa. Para cambiar de panel solo debe dar click al botón que tenga el nombre de la funcionalidad que desee.

Los datos de la aplicación se guardan en la carpeta Data, esta se genera automáticamente en el directorio donde se sitúa el ejecutable.

Figura A.1: Vista primera ejecución.

A.2 Reproductor

La primera funcionalidad que nos encontramos es el reproductor de audio. En este panel el usuario puede reproducir archivos de audio con formato MIDI y WAV. La Figura A.2 muestra este panel en funcionamiento. A continuación, se detalla cada uno de los elementos de la interfaz de arriba a abajo.

-Un campo de entrada para indicar la ruta del audio que se va a reproducir; además, si pulsa el botón Examinar, se abrirá un explorador de archivo para facilitar esta operación.

-Un texto que indica el estado del reproductor.

-Una barra de tiempo que permite al usuario avanzar y retroceder la reproducción.

-Un panel de botones para poder parar, pausar/iniciar y reiniciar la reproducción.

-Un control de volumen que permite subir y bajar la intensidad de la salida del reproductor.

Figura A.2: Panel Reproductor de audio.

A.3 Transformador Audio

La segunda funcionalidad en la lista es el transformador de audio. En este panel el usuario puede transformar un audio con formato WAV a otro con formato MIDI. La Figura A.3 muestra este panel en funcionamiento. A continuación, se detalla cada uno de los elementos de la interfaz de arriba a abajo.

-Un campo de entrada para indicar la ruta del archivo WAV entrante, con posibilidad de usar el explorador de archivos mencionado en el apartado del reproductor.

-Un campo de entrada para indicar la ruta del archivo MIDI saliente, con posibilidad de usar el explorador de archivos.

-Dos campos de entrada para indicar el número de voces y el tempo; además, ambos campos cuentan con botones para facilitar el incremento y decremento de los valores de cada campo.

-Un botón para iniciar/parar la transformación.

-Una barra de progreso donde se muestra el porcentaje realizado de la transformación.

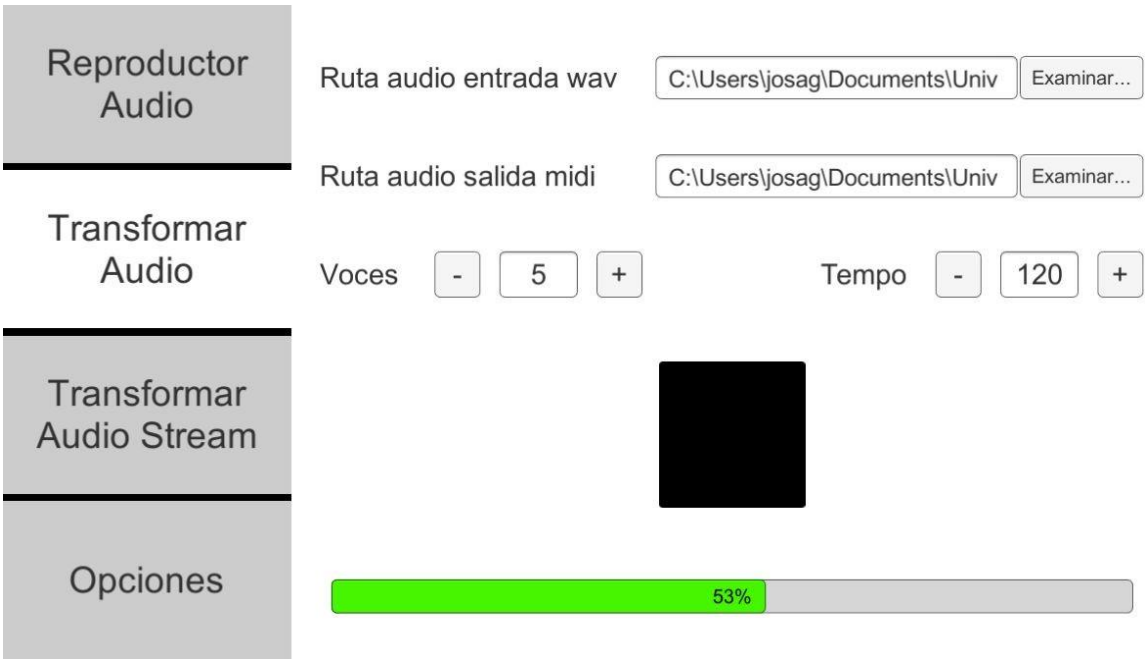

Figura A.3: Panel Transformador Audio.

A.4 Transformador Audio Stream

La tercera funcionalidad a explicar es el transformador de audio stream. Este panel permite al usuario transformar a tiempo real el sonido ambiente captado por el micrófono y mientras que se realiza la reproducción del audio transformado. La Figura A.4 muestra este panel en funcionamiento. A continuación, se detalla cada uno de los elementos de la interfaz de arriba a abajo.

-Dos campos de entrada para indicar el número de voces y el tempo; además, ambos campos cuentan con botones para facilitar el incremento y decremento de los valores de cada campo.

-Un cronógrafo que muestra el tiempo transcurrido desde el inicio de la transformación.

-Un botón para iniciar/parar la transformación.

-Un control de volumen que permite subir y bajar la intensidad de la salida del audio resultante de la transformación.

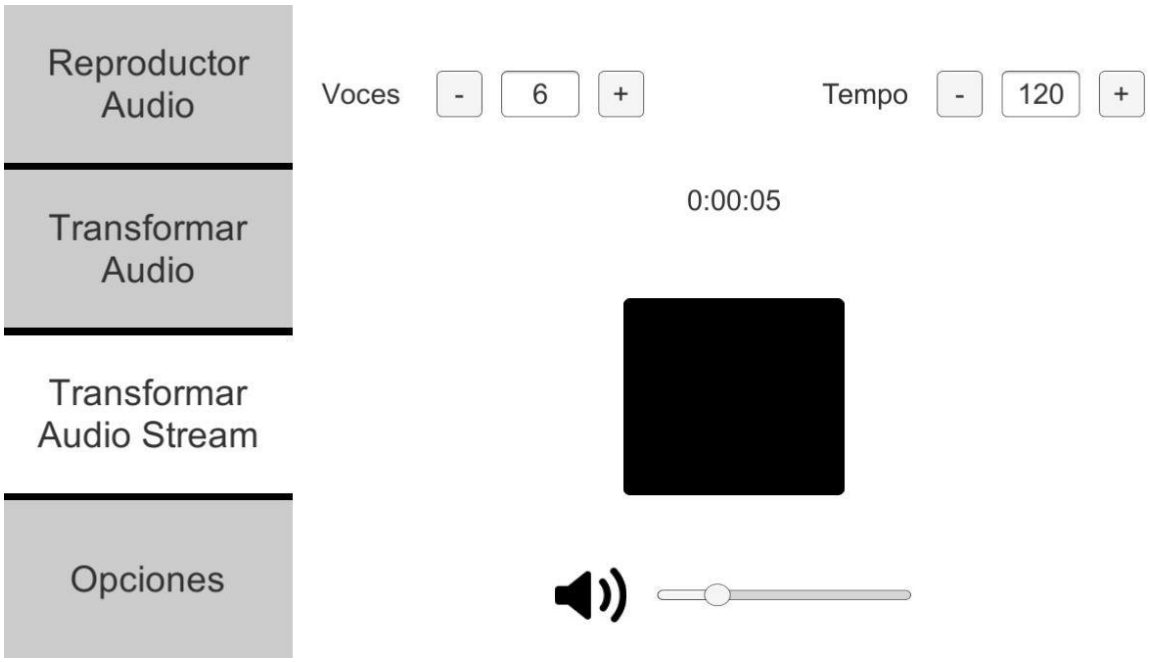

Figura A.3: Panel Transformador Audio Stream.

A.5 Opciones

La cuarta y última funcionalidad disponible es la configuración de la aplicación. En este panel el usuario puede cambiar el color de fondo y modificar la relación entre fuente de audio e instrumento; además en este panel se encuentra el listado de recursos de tercero que emplea el sistema. La Figura A.5 muestra este panel. A continuación, se detalla cada uno de los elementos de la interfaz de arriba a abajo.

-Una lista con botones que permite cambiar el color de fondo de la interfaz.

-Dos listas, la primera muestra la relación entre la clase de fuente posible de un audio y su instrumento asociado, la segunda muestra todos los instrumentos disponibles. Para cambiar de instrumento el usuario debe seleccionar en la tabla

de la izquierda, mediante un click con el botón izquierdo del ratón, la relación que desee cambiar, posteriormente, escoja el instrumento de la tabla de la derecha que quiera asociar con la clase anteriormente seleccionada. También puede escuchar cómo suena cada instrumento manteniendo pulsado cuando haga click con el botón derecho del ratón en cualquier elemento de las dos listas; además, puede usar la rueda del ratón para cambiar la nota que se esté reproduciendo.

-Un control de volumen que permite subir y bajar la intensidad de la nota que se reproduce a la hora de probar los instrumentos de las dos listas mencionadas en el apartado anterior.

-Un botón para poder ver los recursos de terceros empleados en el desarrollo de la aplicación. Cada elemento de la lista de recursos es un enlace hacia la página web de dicho elemento, dicho enlace se activa si se hace click con el botón izquierdo en algún elemento.

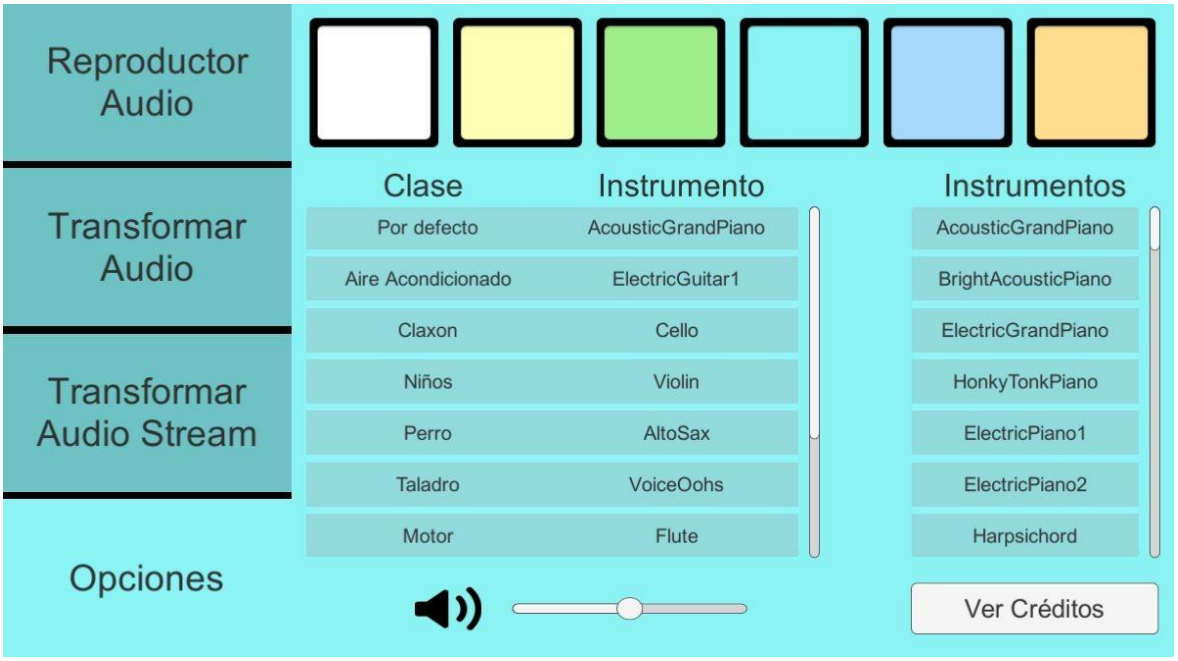

Figura A.5: Panel Opciones.